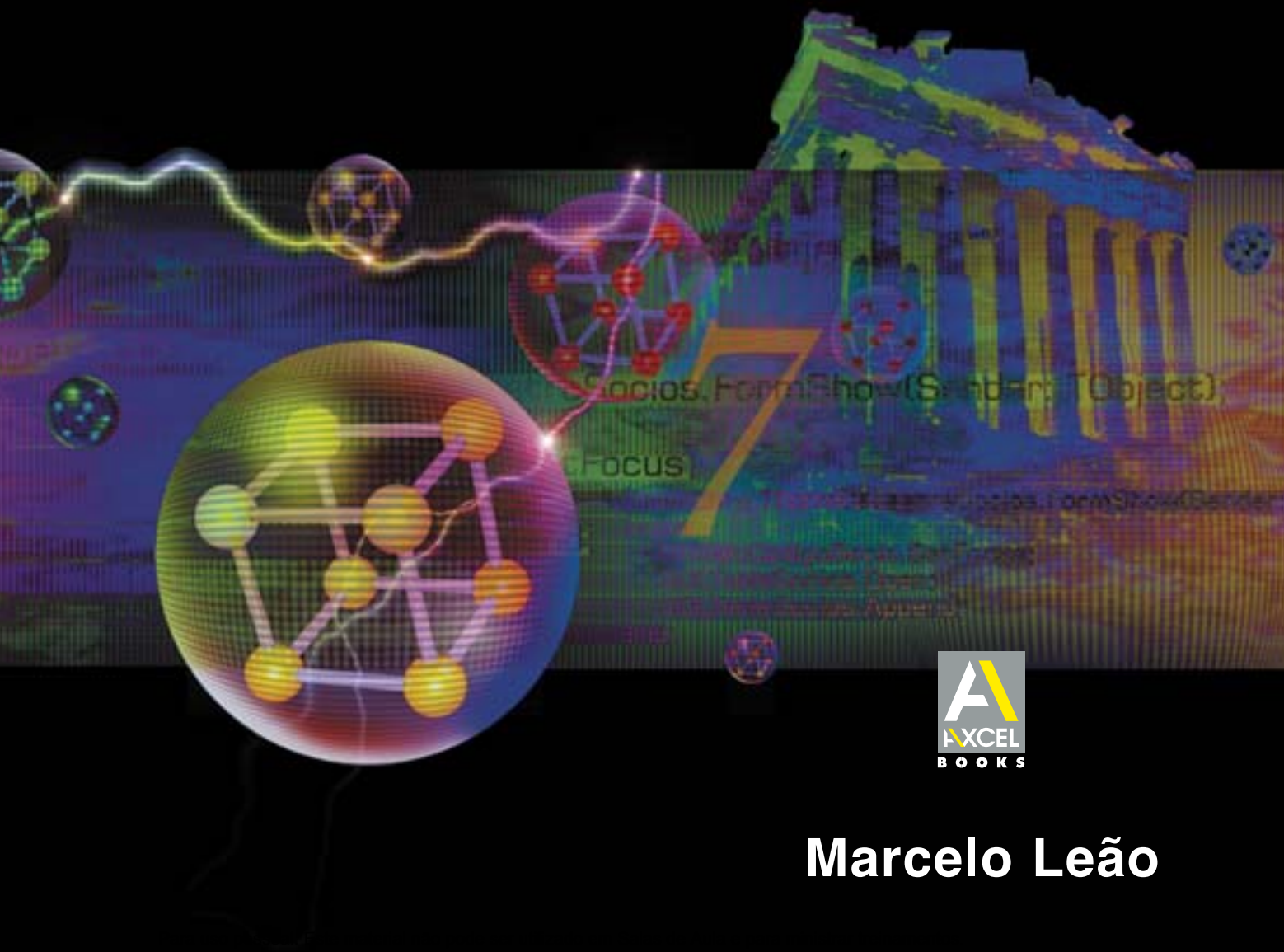
BORLAND®

DELPHI™ 7

CURSO COMPLETO



Marcelo Leão

Editora de Produção: *Gisella Narcisi*
Editor Responsável: *Ricardo Reinprecht*
Projeto Gráfico: *Axcel Books*
Equipe Axcel Books: *Alberto Baptista Garcia, Carlos Alberto Sá Ferreira, Fagner Silva Henrique, Ingo Bertelli*

Borland Delphi 7 Curso Completo
Marcelo Leão
ISBN: *85-7323-184-X*

Axcel Books do Brasil Editora
Av. Paris, 571 – Bonsucesso
21041-020 – Rio de Janeiro – RJ
Tel. (21) 2564-0085 – Fax: (21) 2564-0085
E-mail: editora@axcel.com.br
Visite nossa Home Page
http://www.axcel.com.br
E-mail do Autor: mrleao@rio.com.br

Dedicatória

À minha esposa Beatriz e aos meus filhos Thiago e Lucas, minhas constantes e principais fontes de estímulo e carinho.

Agradecimentos

A todos que sempre me apoiaram ao longo desta dura jornada e a todos que me acompanham no dia-a-dia e, ainda que de forma indireta, colaboraram para que este trabalho fosse concluído.

Aos meus pais Pery e May Leão, pelos exemplos de carinho e dedicação; aos meus irmãos Pery Jr, Ernesto e André pela eterna amizade; aos meus super tios Mara e Nestor Vieira e aos meus super sogros Sérgio e Marly, pelo incondicional apoio; À Beth Ruth Castro da Silveira, pelo apoio e paciência.

Ao competente pessoal da Axcel Books, pelo trabalho sério e competente.

Ao José Eugênio, José Rubens, Mariana Lima, Tertius e a todo pessoal da Borland Latin America, pelo apoio e profissionalismo.

Ao pessoal do Clube Delphi: Gladstone, Luciano, Júlio, Rosângela, Vinícius e demais integrantes, pelo apoio, amizade e incentivo.

Ao meu amigo e conselheiro Professor Sílvio Macieira, amigo de todas as horas, obrigado pela sua amizade!

Aos amigos do CDEM 2002, pela paciência e amizade.

A fim de evitar o risco de cometer injustiças por algum esquecimento, não vou listar a enorme relação de amigos e amigas que sempre me prestigiam com a sua valorosa amizade. Graças a deus, a lista completa de Amigos do Marcelo Leão ocuparia um livro inteiro. Mas vale acrescentar alguns nomes aos listados na edição anterior: Alexander Mazolii, Áureo Pinheiro Ruffier, Edgar Gurgel, Edson Belém, Emerson Moraes, Érico Fagundes Lisboa, Guto Garcia, Luiz Felipe Estrella, Regina Célia, Renato Pitta.

Sobre o Autor

Marcelo Leão é engenheiro formado pelo Instituto Militar de Engenharia (IME) e Mestre em Ciências pela mesma instituição. Atualmente é professor e coordenador do Curso de Graduação em Engenharia do Instituto Militar de Engenharia, professor do Curso de Graduação em Informática da Universidade Estácio de Sá, tendo coordenado o Campus Penha desta universidade, além de ministrar cursos de informática em diversas empresas de treinamento.

Sumário

Parte I: Fundamentos 1

Capítulo 1: Introdução 2

Fundamentos em: Ferramentas de Desenvolvimento 4

As Origens das Ferramentas RAD 4

Capítulo 2: O Ambiente de Desenvolvimento Integrado do Delphi 7 9

Fundamentos em: Composição do Ambiente de Desenvolvimento 10

Apresentação do Ambiente de Desenvolvimento Integrado do Delphi 7 10
Formulários – Os Elementos de Criação da Interface com o Usuário 14
Controles e Componentes 15
Propriedades 16
Métodos 17
Eventos 18
O Object Inspector 18
Alterando o Valor de uma Propriedade no Object Inspector 19
Definindo Procedimentos Associados a Eventos 20
O Conceito de Projeto de uma Aplicação 22
Iniciando um Novo Projeto 28
Salvando o Projeto Recém-criado 29
Fechando um Projeto 31
Manipulando Grupos de Projetos 32
O Code Explorer 35
Desabilitando o Recurso de Ancoragem de Janelas 36
Garantindo a Visibilidade de uma Janela 37

Capítulo 3: Fundamentos da Linguagem Object Pascal 39

Fundamentos em: Estrutura de uma Unidade de Código (Unit) 40

Examinando o Código de uma Unidade de Código (Unit) Gerado Pelo Delphi 7 40
Examinando as Seções de uma Unit 41
Acessando Tipos e Variáveis Definidos em Outras Units 42

Fundamentos em: Declaração de Variáveis 42

O Conceito de Variáveis 42
Atribuindo um Valor a uma Variável 43
Tipos de Dados Predefinidos na Linguagem Object Pascal 43
Tipos de Variáveis Inteiras 43
Tipos de Variáveis Reais 44
Tipos de Variáveis Booleanas 44
Tipos de Variáveis Para Manipulação de Caracteres 44
Tipos de Variáveis Para Manipulação de Arquivos 45
Tipo Genérico de Variáveis 46
Comentários 46
Comentários de uma única linha 46
Comentários de múltiplas linhas 46
Definindo Novos Tipos de Dados 46
Tipos de Dados Enumerados 47
Conjuntos 47
Variáveis Compostas 49
Vetores (Arrays) 49
Operadores Aritméticos 51
Tipos Ordinais 52
Escopo e Tempo de Vida das Variáveis 52
Variáveis Locais 52
Variáveis Globais a uma Unidade de Código 53
Criação de Variáveis Globais a uma Aplicação 53
Adicionando uma Nova Unit ao Projeto Para Armazenamento de Variáveis Globais 53

Fundamentos em: Blocos de Comandos, Estruturas Condicionais e de Repetição 54

Alterando o Fluxo de Execução do Programa 54
Blocos de Comandos 55
Estruturas Condicionais 55
Estrutura Condicional if-then-else 55
Estrutura Condicional Case Of 56
Testes Condicionais 56
Os Operadores Relacionais 56
Estruturas de Repetição 57
Laços For 57
Laços While 58
Laços Repeat 58
Condições Compostas 59
Operadores Lógicos da Linguagem Object Pascal 59
Fundamentos em: Funções e Procedimentos 59
A Estratégia de Dividir Para Conquistar 60
Procedimentos (Procedures) 60
Funções 61
Funções e Procedimentos Para Manipulação de Arquivos Representados por Variáveis 62
Funções e Procedimentos Para Manipulação Direta de Arquivos 63
Fundamentos em: Classes e Objetos 63
Uma Nova (Mas Já Não Tão Nova) Abordagem 64
As Classes e os Objetos 64
O Conceito de Classes 64
Métodos de uma Classe 65
Métodos sem Parâmetros 67
O Objeto Formulário e a Classe TForm 68
O Conceito de Herança de Classes 69
Tipos de Métodos e Campos 70
Procedimentos Associados a Eventos 71
Pare e Reflita Antes de Prosseguir 72
CAPÍTULO 4: PLANEJANDO A SUA APLICAÇÃO 73
Fundamentos em: Planejamento de Aplicações 74
Planejando o seu Trabalho 74
A Importância de um Bom Planejamento 74
Planejando o Nosso Aplicativo-exemplo 75
Padronizando a Nomenclatura dos Componentes 76
Adicionando um Item a uma To-Do Lists 77
Fundamentos em: To-Do Lists 77
Organizando o seu Trabalho com as Ferramentas To-Do List 77
Editando um Item de uma To-Do Lists 78
Excluindo um Item de uma To-Do Lists 79
Configurando as Informações Exibidas em uma To-Do Lists 79
Configurando Como as Informações Devem
Ser Ordenadas em uma To-Do Lists 80
Adicionando um Item a uma To-Do Lists Diretamente no Código-fonte 80
Copiando a Relação de Itens de uma To-Do List 81
Filtrando a Relação de Itens de uma To-Do List 82
CAPÍTULO 5: CRIANDO O FORMULÁRIO PRINCIPAL DA APLICAÇÃO 83
Fundamentos em: Manipulação de Formulários 84
Os Formulários – Elementos Para a Construção da
Interface em Aplicações Desenvolvidas em Delphi 7 84
O Objeto Formulário 84
Propriedades com um Conjunto de Valores Predefinidos 85
Alterando as Propriedades do Objeto Formulário 88
Definindo um Ícone Para o Formulário Principal da Aplicação 89
Inserindo Componentes em um Formulário 90
Inserindo um Componente Para Exibição de Imagens no Formulário Principal 90
Renomeando um Componente 92
Reposicionando um Componente 92
Redimensionando um Componente 93
Alterando a Fonte do Texto Exibido em um Componente 96

Selecionando Vários Componentes Simultaneamente ... 96
Alinhando Componentes ... 97
Analisando o Código Gerado Pelo Delphi ... 99
Testando a sua Aplicação ... 99
Finalizando a Execução do Aplicativo ... 100
Capítulo 6: Projetando um Menu Para a sua Aplicação ... 101
Fundamentos em: Criação de Menus ... 102
Menus – Elementos Indispensáveis ao Formulário Principal de uma Aplicação ... 102
Incluindo um Menu na sua Aplicação ... 102
Acessando o Editor de Menus ... 103
Criando Itens de Menu ... 105
Criando um Separador de Itens em um Menu ... 107
Criando Teclas Aceleradoras Para Itens de Menu ... 108
Criando Outros Itens de Menu ... 108
Incluindo um Menu Pop-up na sua Aplicação ... 109
Criando Itens de Menu em um Menu Pop-up ... 110
Criando Submenus no Menu Pop-up ... 112
Associando Eventos a Itens de Menu ... 114
Definindo Procedimentos Associando Eventos Para Itens de Menu Pop-up ... 115
Capítulo 7: Manipulando Formulários e Caixas de Diálogo ... 117
Fundamentos em: Criação de Caixas de Diálogo ... 118
Caixas de Diálogo – Elementos de Interface
Que Dão Vida ao seu Aplicativo ... 118
Criando uma Caixa de Diálogo de Direitos Autorais ... 118
Personalizando a Caixa de Diálogo de Direitos Autorais ... 119
Exibindo uma Caixa de Diálogo ... 120
O Componente Botão de Comando ... 122
Principais Propriedades do Componente Botão de Comando ... 123
Capítulo 8: Fundamentos do Projeto de Aplicativos de Banco de Dados ... 125
Fundamentos em: Mecanismos de Acesso a Bancos de Dados ... 126
Mecanismos de Acesso a Bancos de Dados ... 126
Fundamentos em: Conceitos Fundamentais Sobre Bancos de Dados ... 127
Conceitos Fundamentais ... 127
Custos ... 128
Planejando seu Banco de Dados ... 128
Capítulo 9: Fundamentos de Bancos de Dados ... 129
Fundamentos em: Criação de Tabelas do Interbase ... 130
Conceitos Fundamentais ... 130
Criando um Banco de Dados no Interbase ... 130
Criando Tabelas com o Database Desktop ... 133
Definindo Nomes Para os Campos dos Registros de uma Tabela ... 136
Definindo Tipos Para os Campos dos Registros de uma Tabela ... 138
Definindo os Tamanhos Para os Campos dos Registros de uma Tabela ... 139
Definindo Campos de Preenchimento Obrigatório ... 141
Criando Índices ... 141
Salvando uma Tabela ... 144
Inserindo Dados em uma Tabela Através do Database Desktop ... 144
Construindo as Demais Tabelas do Aplicativo ... 148
Criando a Tabela Atividades ... 148
Criando Índices Para a Tabela de Atividades ... 149
Salvando a Tabela Atividades ... 149
Incluindo Registros na Tabela de Atividades ... 149
Criando a Tabela de Matrículas ... 150
Criando Índices Para a Tabela de Matrículas ... 150
Capítulo10: Criação de um Formulário Para Manipulação de Tabelas de Bancos de Dados com o DBExpress ... 151
Criação de Formulários Para Acesso a Dados ... 152
Utilizando os Componentes DatasetProvider e ClientDataset ... 159
O Componente DBEdit ... 164

Definindo Máscaras Para os Campos ... 164
Significado dos Caracteres Usados na Propriedade EditMask ... 165
O Componente MaskEdit ... 167
Fazendo a Tecla Enter Funcionar Como Tab ... 167
O Componente DBCombobox ... 168
Propriedades do Controle DBComboBox ... 168
Propriedades do Objeto Items (da Classe TStrings) ... 169
Métodos do Objeto Items (da classe TStrings) ... 169
Destacando o Componente Que Recebe o Foco ... 171

CAPÍTULO 11: CRIAÇÃO DE UM REPOSITÓRIO PARA COMPONENTES DE ACESSO A DADOS ... 173
Criação de Data Modules ... 174
Os Objetos do Tipo DataModule ... 174
Os Objetos de Acesso ao Banco de Dados ... 178
O Componente ClientDataset ... 178
O Componente DataSource ... 180
O Componente SimpleDataset ... 180
Inserindo os Demais Componentes de Acesso ... 181
Para a tabela de Atividades: ... 182
Para a tabela de Matrículas: ... 182
Definindo Índices nos Componentes de Acesso ... 183
Criando um Índice Simples ... 183
Criando um Índice Composto ... 184
Criando Chaves Primárias ... 184

CAPÍTULO 12: CRIAÇÃO DE FORMULÁRIOS PARA CADASTRO DE FORNECEDORES, PRODUTOS E PEDIDOS ... 187
Criação de Formulários de Cadastro ... 188
Criando o Formulário de Cadastro de Atividades ... 188
Criando um Formulário Para Cadastrar Novas Matrículas ... 192
Definindo o Formulário ... 192
Criando o Formulário ... 193
Inserindo os Componentes no Formulário ... 193
Trabalhando com Campos Calculados ... 196

CAPÍTULO 13: CRIANDO FORMULÁRIOS PARA ALTERAÇÃO DE SÓCIOS E ATIVIDADES ... 199
Criação de Formulários Para Alteração de Dados ... 200
O Conceito de Templates de Componentes ... 200
Criando os Templates de Componentes ... 201
Criando o Formulário de Alteração de Sócios ... 202
Criando o Formulário de Alteração de Atividades ... 204
O Componente DBNavigator ... 205

CAPÍTULO 14: CRIANDO FORMULÁRIOS PARA EXCLUSÃO DE SÓCIOS, ATIVIDADES E MATRÍCULAS ... 209
Criação de Formulários Para Exclusão de Dados ... 210
Criando um Template de Formulário ... 210
Criando Formulários a Partir de um Template ... 213
Criando o Formulário de Exclusão de Sócios ... 213
Criando o Formulário de Exclusão de Atividades ... 215
A Linguagem SQL ... 216
Criando o Formulário de Exclusão de Matrículas ... 217

CAPÍTULO 15: CRIANDO FORMULÁRIOS PARA CONSULTA DE SÓCIOS, ATIVIDADES E MATRÍCULAS ... 223
Criação de Formulários Para Consulta de Dados ... 224
Criando um Formulário Para a Consulta de Dados dos Sócios ... 224
Criando um formulário Para a Consulta de Dados das Atividades ... 225
Criando um Formulário Para a Consulta de Dados das Matrículas ... 226

CAPÍTULO 16: CRIANDO ROTINAS DE BACKUP E RESTAURAÇÃO ... 227
Cópia de Arquivos ... 228
Conceitos Fundamentais ... 228
Criando um Formulário de Backup ... 228

PARTE II: KNOW-HOW ... 233

CAPÍTULO 17: CRIAÇÃO DE RELATÓRIOS COM O RAVE REPORTS ... 235
Know-How em: Criação de Relatórios com o Rave Reports ... 236

Introdução ... 236
O Componente RvProject ... 236
Criando um Relatório de Sócios ... 236
Criando um Relatório de Atividades ... 242
Criando um Relatório de Matrículas ... 244
CAPÍTULO 18: INCORPORANDO O RECURSO DE HELP ON-LINE À NOSSA APLICAÇÃO ... 247
Know-How em: Criação de Arquivos de Help ... 248
Criando um Arquivo de Help ... 248
Definindo uma Página de Índice ... 248
Criando um Arquivo RTF ... 249
Criando uma Página Para Cada Tópico ... 250
Criando Strings de Contexto ... 250
Criando uma Palavra-chave Para um Tópico ... 252
Criando um Título Para um Tópico ... 252
Associando um Número de Página a um Tópico ... 253
Estabelecendo a Conexão Entre Tópicos ... 254
Criando um Arquivo de Projeto de Help ... 256
Criando e Compilando o Arquivo de Projeto de Help com o Microsoft Help Workshop ... 257
Associando o Arquivo de Help à sua Aplicação ... 260
Associando um Componente a um Tópico do Arquivo de Help ... 261
Exibindo o Arquivo de Help em Resposta a um Item de Menu ... 262
CAPÍTULO 19: PROGRAMAÇÃO ORIENTADA A OBJETOS EM DELPHI 7 ... 263
KNOW-HOW em: Programação Procedural em Linguagem Object Pascal ... 264
Apresentação do Problema ... 265
Utilizando o Ambiente do Delphi 7 Para o Pascal Procedural ... 266
KNOW-HOW em: Fundamentos da Programação Orientada a Objetos ... 269
A Filosofia da Programação Orientada a Objetos ... 269
Análise do Código-fonte ... 271
A Implementação de uma Classe ... 272
Análise do Código-fonte ... 274
KNOW-HOW em: Herança de Classes ... 274
O Conceito de Herança de Classes ... 275
Métodos Construtores ... 277
Métodos Destrutores ... 278
Visibilidade dos Campos e Métodos de uma Classe ... 279
Campos e Métodos Públicos (public) ... 279
Campos e Métodos Privados (private) ... 280
Campos e Métodos Protegidos (protected) ... 280
Sobreposição de Métodos ... 281
Métodos Estáticos, Virtuais e Dinâmicos ... 282
Mas e os Métodos Dinâmicos? O Que Significam? ... 284
Métodos Abstratos ... 284
Métodos de Classe ... 285
Propriedades ... 285
Referência: a Classe TObject ... 287
Principais Métodos da Classe TObject ... 288
KNOW-HOW em: Conversão de Tipos ... 294
O Conceito de Conversão de Tipos ... 294
O Operador Is ... 294
O Operador As ... 295
Conversão Explícita Entre Tipos ... 295
O Identificador Self ... 296
KNOW-HOW em: Tratamento de Exceções ... 296
Técnica ... 296
O Conceito de Exceções ... 296
Referência: a Classe Exception ... 298
Definição da Classe Exception ... 298
Campos Internos da Classe Exception ... 298
Métodos Públicos da Classe Exception ... 299
Propriedades da Classe Exception ... 302
Classes Derivadas por Herança da Classe Exception ... 302

O Mecanismo Utilizado na Geração de Exceções ... 302
KNOW-HOW em: Manipulação de Listas de Objetos ... 304
Apresentação do Problema ... 304
Referência: a Classe TList ... 305
Definição da Classe TList ... 305
Propriedades da Classe TList ... 305
Principais Métodos da Classe TList ... 306
Exemplo de Utilização ... 309
Definição da Interface ... 309
Codificação do Exemplo ... 311
KNOW-HOW em: Sobrecarga de Métodos ... 322
O Conceito de Sobrecarga de Métodos ... 323
KNOW-HOW em: Definição de Parâmetros Default Para uma Função ou Procedimento ... 324
O Conceito de Parâmetro Default Para uma Função ou Procedimento ... 324
CAPÍTULO 20: O CONCEITO DE COMPONENTES ... 327
KNOW-HOW em: Definição de Componentes ... 328
O Conceito de Componentes ... 329
Exibindo uma Propriedade no Object Inspector ... 331
O Sistema de Mensagens do Windows ... 331
A Classe TCanvas ... 335
Principais Propriedades da Classe TCanvas ... 336
Principais Métodos da Classe TCanvas ... 337
Principais Propriedades da Classe TPen ... 341
Principais Propriedades da Classe TBrush ... 343
O Componente Shape ... 343
CAPÍTULO 21: CRIAÇÃO DE COMPONENTES ... 347
KNOW-HOW em: Criação de Componentes ... 348
Apresentação do Problema ... 348
Criando o Esqueleto do Novo Componente ... 350
Definição de Novas Propriedades ... 352
Criando uma Nova Propriedade ... 353
Instalando o Novo Componente ... 354
Sobrecarregando o Método Construtor da Classe Ancestral do Componente ... 356
Redefinindo Métodos da Classe-base ... 358
Definindo um Novo Evento Para o Componente ... 360
O Tipo TNotifyEvent ... 361
Definindo um Método de Leitura Para uma Propriedade ... 362
KNOW-HOW em: Criação de Componentes Associados a Bancos de Dados ... 364
Apresentação do Problema ... 365
Criando o Esqueleto do Novo Componente ... 365
Definindo as Novas Propriedades Para o Componente ... 366
Refletindo Alterações Feitas no Campo ... 369
Refletindo Alterações Feitas no Componente ... 371
Notificando o Componente da Remoção de um DataSource ... 374
Criando uma Propriedade que Permita Tratar a Tecla Enter Como Tab ... 376
KNOW-HOW em: Criação de Controles ActiveX ... 379
Apresentação do Problema ... 379
Convertendo o Componente NumEdit em um Controle ActiveX ... 379
CAPÍTULO 22: MECANISMOS DE ACESSO A BANCO DE DADOS ... 393
KNOW-HOW em: Fundamentos dos Mecanismos de Acesso a Bancos de Dados ... 394
Os Mecanismos de Acesso a Bancos de Dados ... 394
KNOW-HOW em: Classes Fundamentais de Acesso a Bancos de Dados – A Classe TDataset ... 395
A Classe TDataSet ... 395
Principais Propriedades da Classe TDataSet ... 397
Principais Métodos da Classe TDataSet ... 401
Principais Eventos da Classe TDataSet ... 406
A Classe TCustomConnection ... 411
Principais Propriedades da Classe TCustomConnection ... 411
Principais Métodos da Classe TCustomConnection ... 411
Principais Eventos da Classe TCustomConnection ... 413

CAPÍTULO 23: BANCO DE DADOS – COMPONENTES DE ACESSO VIA BDE 415
KNOW-HOW em: CLasses Fundamentais de Acesso a Bancos de Dados via BDE – As Classes TBDEDataset e TDBDataset 416
A Classe TBDEDataSet 416
Principais Propriedades da Classe TBDEDataSet 417
Principais Métodos da Classe TBDEDataSet 417
A Classe TDBDataSet 418
Principais Propriedades da Classe TDBDataSet 418
A Classe TDatabase 419
Principais Propriedades da Classe TDatabase 419
Principais Métodos do Componente Database 421
A Classe TSession 422
Principais Propriedades da Classe TSession 422
Principais Métodos da Classe TSession 424
Eventos do Componente Session 427
KNOW-HOW em: CLasses de Acesso Direto a Bancos de Dados via BDE – As Classes TTable e TQuery 427
A Classe TTable 428
Principais Propriedades da Classe TTable 428
Principais Métodos da Classe TTable 430
A Classe TQuery 434
Principais Propriedades da Classe TQuery 435
Principais Métodos da Classe TQuery 436
A Classe TUpdateSQL 437
Principais Propriedades da Classe TUpdateSQL 437
Principais Métodos da Classe TUpdateSQL 438
Exemplos de Aplicação 438
Indexação de Tabelas Acessadas Pelo Componente Table 438
Exemplo de Aplicação 439
Filtrando os Registros de uma Tabela Acessada Pelo Componente Table 442
Estabelecendo um Relacionamento Entre Tabelas Representadas Pelo Componente Table 454
Pesquisando Registros em Tabelas Representadas Pelo Componente Table 458
Criação de Tabelas em Run-time 463
Componentes e Métodos de Navegação 473
Tradução da Mensagem Delete Record do Componente Table 480
Exemplo de Utilização do Componente TSession 481
Consulta a Bancos de Dados via Declarações SQL Definidas em Run-Time 487
Utilização de Parâmetros em Declarações SQL 491
Utilização do Recurso de Cached Updates 495
Exemplo de Utilização do Componente TUpdateSQL 500
KNOW-HOW em: Aplicação de Senhas a Tabelas do Tipo Paradox 506
Definindo Senhas Para uma Tabela do Tipo Paradox 507
Definindo Senhas Auxiliares Para uma Tabela do Tipo Paradox 508
Inibindo a Exibição da Caixa de Diálogo Enter Password Durante a Execução do seu Aplicativo 509
Protegendo Sua Aplicação Mediante Definição de uma Senha 509
CAPÍTULO 24: BANCO DE DADOS – COMPONENTES DE ACESSO VIA ADO 511
KNOW-HOW em: CLasses Fundamentais de Acesso a Bancos de Dados via ADO – A Classe TCustomADODataset e os Componentes TADOConnection, TRDSConnection e TADODataset e TADOCommand 512
O Componente TADOConnection 512
Principais Propriedades da Classe TADOConnection 512
Principais Métodos da Classe TADOConnection 514
Principais Eventos da Classe TADOConnection 516
O Componente TRDSConnection 517
Principais Propriedades da Classe TRDSConnection 517
Principais Métodos da Classe TRDSConnection 517
Principais Eventos da Classe TRDSConnection 517
A Classe TCustomADODataset 518
Principais Propriedades da Classe TCustomADODataSet 518
Principais Métodos da Classe TCustomADODataset 521

Principais Eventos da Classe TCustomADODataset ... 524
A Classe TADOCommand ... 525
Principais Propriedades da Classe TADOCommand ... 525
Principais Métodos da Classe TADOCommand ... 526
Principais Eventos da Classe TADOCommand ... 527
A Classe TADODataset ... 527
Principais Propriedades da Classe TADODataSet ... 527
Principais Métodos da Classe TADODataset ... 528
Principais Eventos da Classe TADODataset ... 528
KNOW-HOW em: Classes de Acesso Direto a Bancos de dados via BDE – As Classes TADOTable e TADOQuery ... 528
A Classe TADOTable ... 529
Principais Propriedades da Classe TADOTable ... 529
Principais Métodos da Classe TADOTable ... 530
Principais Eventos da Classe TADOTable ... 530
A Classe TADOQuery ... 530
Principais Propriedades da Classe TADOQuery ... 530
Principais Métodos da Classe TADOQuery ... 530
Principais Eventos da Classe TADOQuery ... 531
Exemplos de Aplicação ... 531
Estabelecendo uma Conexão a Bancos de Dados do MS Acess com o Componente ADOConnection ... 531
Acessando tabelas do Access com o Componente ADOComando ... 534
Acessando Tabelas do Access com o Componente ADODataset ... 534
Acessando tabelas do Access com o Componente ADOTable ... 535
Acessando tabelas do Access com o Componente ADOQuery ... 536
Diferenças na Utilização dos Componentes Table x ADOTable, e Query x ADOQuery ... 536
Definição do Índice Corrente ... 536
Pesquisando Registros em Tabelas Representadas Pelo Componente ADOTable ... 537
Criação de Tabelas em Run-time ... 537
Componentes e Métodos de Navegação ... 537
Utilização de Parâmetros em Declarações SQL ... 537
CAPÍTULO 25: BANCO DE DADOS – COMPONENTES DE ACESSO VIA DBEXPRESS ... 539
KNOW-HOW em: Componentes de Acesso a Bancos de Dados via DBExpress ... 540
O Componente TSQLConnection ... 540
Principais Propriedades da Classe TSQLConnection ... 540
Principais Métodos da Classe TSQLConnection ... 542
Principais Eventos da Classe TSQLConnection ... 543
A Classe TCustomSQLDataset ... 543
Principais Propriedades da Classe TCustomSQLDataSet ... 544
Principais Eventos da Classe TCustomSQLDataset ... 544
A Classe TSQLDataSet ... 544
Principais Propriedades da Classe TSQLDataSet ... 545
KNOW-HOW em: Classes de Acesso Direto a Bancos de Dados via DBExpress – As Classes TSQLTable, TSQLQuery e TSQLStoredProc ... 545
A Classe TSQLTable ... 546
Principais Propriedades da Classe TSQLTable ... 546
Principais Métodos da Classe TSQLTable ... 547
Principais Eventos da Classe TSQLTable ... 547
A Classe TSQLQuery ... 547
Principais Propriedades da Classe TSQLQuery ... 547
Principais Métodos da Classe TSQLQuery ... 547
Principais Eventos da Classe TSQLQuery ... 548
A Classe TSimpleDataset ... 548
Principais Propriedades da Classe TSimpleDataset ... 548
Principais Métodos da Classe TSimpleDataset ... 549
Principais Eventos da Classe TSimpleDataset ... 549
CAPÍTULO 26: BANCO DE DADOS – COMPONENTES DE ACESSO VIA INTERBASE EXPRESS ... 551
KNOW-HOW em: Componentes de Acesso a Bancos de Dados via Interbase Express ... 552

O Componente TIBDatabase 552
Principais Propriedades da Classe TIBDatabase 552
Principais Métodos da Classe TIBDatabase 554
Principais Eventos da Classe TIBDatabase 555
O Componente TIBTransaction 555
Principais Propriedades da Classe TIBTransaction 555
Principais Métodos da Classe TIBTransaction 556
Principais Eventos da Classe TIBTransaction 558
A Classe TIBCustomDataset 558
Principais Propriedades da Classe TIBCustomDataset 559
Principais Métodos da Classe TIBCustomDataset 560
Principais Eventos da Classe TIBCustomDataset 561
A Classe TIBDataSet 561
Principais Propriedades da Classe TIBDataSet 561
Principais Métodos da Classe TIBDataset 562
Principais Eventos da Classe TIBDataset 562
KNOW-HOW em: Classes de Acesso Direto a Bancos de dados via Interbase Express – As Classes TIBTable, TIBQuery e TIBUpdateSQL 563
A Classe TIBTable 563
Principais Propriedades da Classe TIBTable 563
Principais Métodos da Classe TIBTable 565
Principais Eventos da Classe TIBTable 567
A Classe TIBQuery 567
Principais Propriedades da Classe TIBQuery 567
Principais Métodos da Classe TIBQuery 568
Principais Eventos da Classe TIBQuery 568
A Classe TIBUpdateSQL 568
Principais Propriedades da Classe TIBUpdateSQL 569
Principais Métodos da Classe TIBUpdateSQL 569
CAPÍTULO 27: BANCOS DE DADOS CLIENTE/SERVIDOR 571
KNOW-HOW em: Fundamentos dos Bancos de Dados Cliente/Servidor 572
Apresentação do Problema 572
O Administrador do Servidor de Banco de Dados 572
Cadastrando um Novo Usuário 574
Alterando os Dados de um Usuário Já Cadastrado 575
Removendo um Usuário Cadastrado 575
O Utilitário Interactive SQL 576
Criando um Banco de Dados no Interbase a Partir do Utilitário Interactive SQL 576
Conectando-se a um Banco de Dados do Interbase 578
Tipos de Dados Definidos Pelo Interbase 578
Criando uma Tabela no Interbase 579
Aplicando Restrições aos Campos de uma Tabela 580
Removendo uma Tabela do Banco de Dados 580
Criando Índices em uma Tabela 580
Concedendo Privilégios a um Outro Usuário ou Banco de Dados 581
Removendo Privilégios 582
Visões (Views) 582
O Conceito de Transações 583
Incluindo Registros com o Comando INSERT 583
Atualizando Registros em uma Tabela 584
Removendo Registros de uma Tabela 584
Ordenando os Registros de uma Tabela 585
TRIGGERS 585
STORED PROCEDURES (Procedimentos Armazenados) 586
Linguagem de Codificação do Interbase 587
Declaração de Variáveis no Interbase 587
Atribuição de Valores a Variáveis no Interbase 588
Definição de Comentários no Interbase 588
Estruturas Condicionais do Interbase 588
Estruturas de Repetição do Interbase 588

Criação de Novos Tipos no Interbase ... 589
Metadados de um Banco de Dados ... 589
Concatenando Dados Provenientes de Várias Tabelas ... 590
Criando um Backup de um Banco de Dados do Interbase ... 591
Recuperando um Banco de Dados a Partir de um Backup ... 591
Criando um Alias Para um Banco de Dados do Interbase ... 592
CAPÍTULO 28: ACESSANDO BANCOS DE DADOS CLIENTE/SERVIDOR ... 593
KNOW-HOW em: Acesso a Bancos de Dados Cliente/Servidor ... 594
Apresentação do Problema ... 594
Acessando Banco de Dados Cliente-Servidor do Interbase em uma Aplicação Delphi via BDE ... 594
Principais Propriedades do Componente StoredProc ... 595
Principais Métodos do Componente StoredProc ... 596
Principais Eventos do Componente IBoredProc ... 597
Exemplo de Aplicação ... 597
Acessando Banco de Dados do Interbase em uma Aplicação Delphi via Interbase Express ... 600
Principais Propriedades do Componente IBSToredProc ... 601
Principais Métodos do Componente IBStoredProc ... 601
Principais Eventos do Componente IBStoredProc ... 602
Exemplo de Aplicação ... 603
Acessando Banco de Dados do Interbase em uma Aplicação Delphi via DBExpress ... 605
Principais Propriedades do Componente SQLSToredProc ... 605
Principais Métodos do Componente SQLStoredProc ... 606
Principais Eventos do Componente SQLStoredProc ... 606
Exemplo de Aplicação ... 606
CAPÍTULO 29: PROGRAMAÇÃO GRÁFICA ... 609
KNOW-HOW em: Definição de Desenhos em Run-Time ... 610
A Classe TCanvas ... 610
O Componente Shape ... 611
Desenhando em um Formulário ... 611
KNOW-HOW em: Definição de Desenhos de Forma Interativa ... 620
Desenhando de Forma Interativa ... 621
A Classe TBitmap ... 626
Modos de Mapeamento ... 630
Funções Para Transformação de Coordenadas ... 632
Função SetWindowExtEx ... 632
Função SetViewportExtEx ... 633
CAPÍTULO 30: TÉCNICAS DE IMPRESSÃO ... 635
KNOW-HOW em: Impressão Direta no Windows ... 636
Principais Propriedades da Classe TPrinter ... 637
Aborted ... 637
Canvas ... 637
Copies ... 637
Fonts ... 637
Orientation ... 637
PageHeight ... 637
PageNumber ... 637
PageWidth ... 637
PrinterIndex ... 638
Printers ... 638
Printing ... 638
Title ... 638
Principais Métodos da Classe TPrinter ... 638
BeginDoc ... 638
EndDoc ... 638
NewPage ... 639
Exemplo de Utilização da Classe TPrinter ... 639
Criando a Interface da Aplicação ... 639
Codificando a Aplicação ... 641
Imprimindo o Conteúdo Exibido por um Componente Memo ... 643
Criando a Interface da Aplicação ... 643
Codificando a Aplicação ... 643

CAPÍTULO 31: CRIAÇÃO DE DLLS 649
KNOW-HOW em: Criação de DLLs 650
Introdução 650
Procedimentos Básicos Necessários à Criação de uma DLL em Delphi 650
Chamando uma DLL a Partir de Outra Aplicação 652
O Formulário Principal da Aplicação 653
Arquivo de Código Associado ao Formulário 653
Exibindo Formulários a Partir de uma DLL 654
Carregamento Explícito de uma DLL 656
CAPÍTULO 32: MANIPULAÇÃO DE ARQUIVOS, STRINGS E FONTES EM DELPHI 657
KNOW-HOW em: Manipulação de Arquivos 658
Manipulação Direta de Arquivos Associados a uma Variável 658
KNOW-HOW em: Manipulação de Strings 662
Principais Funções Para a Manipulação de Strings 663
Principais Funções Para a Manipulação de Strings de Terminação Nula 670
StrLower 674
StrNew 675
StrPCopy 675
KNOW-HOW em: Manipulação de Listas de Strings 676
A Classe TStrings 676
Referência: A Classe TStrings 676
Definição da Classe TStrings 676
Propriedades da Classe TStrings 677
Métodos da Classe TStrings 679
Componentes que Definem Propriedades Como Objetos da Classe TStrings 682
KNOW-HOW em: Manipulação de Fontes 683
Referência: a Classe TFont 683
Definição da Classe TFont 683
Propriedades da Classe TFont 684
Exemplo de Utilização das Classes TFont e TStrings Para Manipulação de Arquivos Texto ASCII 685
Criando a Interface da Aplicação 685
Codificando a Aplicação 687
Exemplo de Utilização das Classes TFont e TStrings Para Manipulação de Arquivos RTF 696
Criando a Interface da Aplicação 696
Codificando a Aplicação 698
CAPÍTULO 33: MANIPULAÇÃO DE THREADS EM DELPHI 705
KNOW-HOW em: Threads 706
O Conceito de Threads 706
Unidade de Código Associada: 707
A Classe TThread 708
Implementando a Classe TPBThread 711
Redefinindo o Código da Unit Associada ao Formulário 712
CAPÍTULO 34: IMPLEMENTAÇÃO DA TECNOLOGIA COM EM DELPHI 715
KNOW-HOW em: Tecnologia COM 716
A Tecnologia COM 716
Adicionando e Implementando um Método à Interface IUtilitario 720
Compilando e Registrando o Objeto COM no seu Sistema 725
Criando uma Aplicação que Utilize o Objeto COM 725
A Tecnologia OLE 726
CAPÍTULO 35: APLICAÇÕES MULTICAMADAS 727
KNOW-HOW em: Aplicações Multicamadas 728
Apresentando a Tecnologia 728
A Camada de Armazenamento de Informações 728
A Camada Intermediária – a Camada Servidora 728
Criando a Camada de Interface com o Usuário (a aplicação-cliente) 731
CAPÍTULO 36: TÉCNICAS ÚTEIS PARA A CRIAÇÃO DA INTERFACE COM O USUÁRIO 737
KNOW-HOW em: Parametrização de Strings de Auxílio 738
Utilização das Strings de Auxílio (Hints) 738
Alterando a Cor de Fundo do Texto Exibido na String de Auxílio 738

Alterando o Tempo de Início e Término de Exibição da String de Auxílio ... 739
Alterando o Tempo de Exibição Entre Strings de Auxílio Distintas ... 739
Exibindo uma String de Auxílio Composta por Várias Linhas ... 739
KNOW-HOW em: Múltiplas Instâncias ... 740
Apresentação do Problema ... 740
KNOW-HOW em: Reinicialização do sistema a partir de uma aplicação ... 741
Apresentação do Problema ... 741
KNOW-HOW em: Manipulação da data e hora do sistema ... 741
O Tipo TDateTime ... 742
Obtendo a Data e Hora do Sistema ... 742
Convertendo um Valor do Tipo Data/Hora em uma String ... 743
Convertendo uma String em um Valor do Tipo Data/Hora ... 743
Obtendo o Dia da Semana Correspondente a uma Data ... 743
Funções Especiais de Conversão de Data/Hora ... 743
KNOW-HOW em: Personalização de Formulários com a Definição de um Pano de Fundo ... 744
Inserindo um Pano de Fundo em um Formulário ... 744
KNOW-HOW em: Desenvolvimento de Aplicações MDI ... 745
Criando Aplicações MDI ... 745
Criando a Janela Principal de uma Aplicação MDI ... 746
Criando uma Janela-filha de uma Aplicação MDI ... 746
Destruindo Uma Janela-Filha de uma Aplicação MDI ... 746
Organizando a Exibição das Janelas-filhas ... 747
Mesclando Menus ... 747
Codificando a Aplicação ... 748
KNOW-HOW em: Operações de Drag-drop em Componentes ... 751
Apresentação do Problema ... 751
Descrição das Técnicas de Drag & Drop ... 751
Exemplo de Utilização ... 752
Definição da Interface ... 752
Codificação do Exemplo ... 754
CAPÍTULO 37: INTERNACIONALIZAÇÃO DE APLICATIVOS CRIADOS COM O DELPHI ... 757
KNOW-HOW em: Internacionalização de Aplicativos ... 758
O Ambiente Integrado de Tradução do Delphi 7 ... 758
Incorporando os Recusos do Ambiente Integrado de Tradução ao seu Projeto de Aplicativo ... 758
Traduzindo Constantes e Expressões ... 763
Definindo o Idioma Corrente ... 765
Utilizando o Translation Repository ... 766
CAPÍTULO 38: CRIANDO APLICAÇÕES PARA A INTERNET ... 769
KNOW-HOW em: Desenvolvimento de Aplicações CGI com WebBroker ... 770
Procedimentos Básicos Necessários à Criação de Aplicações CGI ... 770
Uma Aplicação CGI Bastante Elementar ... 771
Exibindo a Data e a Hora do Sistema em uma Página HTML ... 776
Respondendo a Entrada de Dados de Formulários HTML ... 777
O Componente PageProducer ... 780
O Componente DataSetTableProducer ... 782
O Componente QueryTableProducer ... 788
KNOW-HOW em: Desenvolvimento de Aplicações CGI com WebSnap ... 791
Criando o Módulo Principal de uma Aplicação WebSnap ... 792
Adicionando um Grid Para Exibição dos Registros ... 796
Adicionando Botões Para a Edição dos Registros do Grid ... 798
Criando uma Página com um Formulário Para Edição dos Registros ... 799
Conectando as Páginas ... 801
Testando a Aplicação ... 801
Tratando Erros ... 803
Criando uma Página de Login ... 803
Definindo as Páginas que Requerem Login ... 805
Definindo Direitos de Acesso a Usuários ... 805
KNOW-HOW em: Desenvolvimento de Aplicações CGI com Intraweb ... 806
Fundamentos da Tecnologia IntraWeb ... 807
Criando a Aplicação ... 807

Criando o Formulário Principal da Aplicação 807
Criando o Formulário de Cadastro 811
Criando o Formulário de Alteração 813
Criando o Formulário de Exclusão 814
Testando a Aplicação 816
Parte III: Referência 819
Capítulo 39: Funções Matemáticas Disponíveis em Delphi 820
Capítulo 40: Classes, Controles e Componentes 829
Capítulo 41: Propriedades 977
Capítulo 42: Métodos 1229
Capítulo 43: Eventos 1339
Índice Remissivo 1383

Parte I

Fundamentos

Capítulo 1

Introdução

Neste capítulo será feita uma breve apresentação dos fatos que antecederam o surgimento das ferramentas de desenvolvimento RAD, dentre as quais se destaca o próprio Delphi, e da importância dessas ferramentas no mercado desenvolvedor.

É importante destacar que o Delphi, agora na versão 7, não é mais uma simples ferramenta para desenvolvimento de aplicações com interface gráfica e baseada na linguagem Object Pascal. O Delphi 7 é, na realidade, um ambiente de desenvolvimento que integra diversas tecnologias.

Fundamentos em: Ferramentas de Desenvolvimento

PRÉ-REQUISITOS

- Noções básicas de desenvolvimento de software. Experiência na utilização do sistema operacional Windows.

METODOLOGIA

- Apresentação do problema: A escolha de uma ferramenta capaz de acelerar o processo de desenvolvimento de software.

As Origens das Ferramentas RAD

Nos últimos anos, o surgimento dos sistemas operacionais com interface gráfica fez com que as tarefas relacionadas ao desenvolvimento de softwares comerciais sofressem transformações radicais.

Inicialmente, o desenvolvimento de sistemas para o ambiente Windows requeria a utilização da linguagem C – na qual estão implementadas as funções da API do Windows. O desenvolvimento de uma aplicação extremamente simples, que exibisse apenas uma janela com alguma mensagem estática, requeria cerca de dezenas de linhas de código em linguagem C. O desenvolvimento de aplicações mais complexas, por sua vez, exigia centenas (se não milhares) de linhas de código em linguagem C – apenas para criar os elementos de interface com o usuário.

Dessa maneira, o desenvolvedor levava um tempo enorme apenas codificando a interface do sistema – cuja finalidade é simplesmente obter informações do usuário ou exibir informações referentes ao resultado de algum processamento –, em vez de se dedicar mais ao código associado a um sistema específico.

Considere, por exemplo, a seguinte situação: um desenvolvedor foi encarregado de criar um sistema para controle de estoque de uma empresa. Não seria mais lógico se dedicar mais à codificação das rotinas destinadas a solucionar os problemas intrínsecos ao sistema do que se preocupar em criar janelas e botões de comando? Evidentemente que sim!

Paralelamente ao problema decorrente da extensa codificação necessária à criação da interface, a linguagem C também não é uma linguagem de programação orientada a objetos – característica desejada às linguagens modernas – o que levaria a uma mudança radical nas técnicas de modelagem e desenvolvimento de sistemas. Além disso, muitos desenvolvedores consideravam a linguagem C extremamente difícil em comparação com o Basic e o Pascal (do qual se originou o Delphi), por exemplo.

Com o surgimento de ambientes de desenvolvimento baseados na linguagem C++ (uma das primeiras linguagens a suportar as características da programação orientada a objetos), como o Borland C++ e o Microsoft Visual C++, as técnicas de programação orientada a objetos passaram a ser incorporadas com mais facilidade

no desenvolvimento dos novos sistemas. A linguagem C++, por ser uma extensão da linguagem C (às vezes denominada por alguns autores como "C com Classes"), podia acessar diretamente as funções da API do Windows e já oferecia uma série de classes que definiam os principais elementos de interface. Continuava, no entanto, a existir uma certa "distância" entre a codificação do aplicativo e a criação da interface com o usuário. A interface com o usuário era desenhada em um editor de recursos e a sua associação ao código era feita de forma indireta, em uma "tabela de respostas a eventos" definida no código da aplicação.

Essa lacuna começou a ser preenchida com o surgimento das primeiras ferramentas para desenvolvimento rápido de aplicativos, também denominadas ferramentas RAD (Rapid Application Development). Essas ferramentas permitiam associar, de maneira simples e rápida, um elemento de interface e o código da aplicação.

Uma das primeiras ferramentas a adotar o conceito RAD foi o Visual Basic for Windows, da Microsoft. Ao adotar uma linguagem simples como o Basic e facilitar a associação entre código e elementos de interface (por meio da incorporação das técnicas de programação orientada a eventos), o Visual Basic reuniu os requisitos básicos para se tornar, inicialmente, uma das ferramentas de desenvolvimento de maior sucesso do mercado.

Havia, no entanto, algumas limitações. Uma aplicação desenvolvida em Visual Basic requeria a utilização de algumas DLLs cuja distribuição era indesejável, e o desempenho dos aplicativos gerados não era tão bom quanto o das aplicações desenvolvidas em C++. Além disso, o Visual Basic não suportava os requisitos de uma linguagem de programação orientada a objetos (o que, no entanto, já ocorre a partir da versão do produto para a plataforma .NET, que incorporou profundas modificações na linguagem).

Os desenvolvedores Windows passaram então a ser obrigados a optar entre desempenho final e velocidade de desenvolvimento. Aqueles que optassem pelos ambientes de desenvolvimento baseados na linguagem C++ ganhavam no desempenho das aplicações geradas, mas necessitavam de prazos superiores aos daqueles que optavam pelo Visual Basic.

Foi então que a Borland surpreendeu o mercado com o lançamento do Borland Delphi (ainda na versão 1.0 – para desenvolvimento de aplicações para o Windows 3.x) – uma ferramenta que aliava a facilidade do Visual Basic ao poder da linguagem Object Pascal, cujo compilador apresentava o mesmo desempenho do Borland C++.

O Delphi apresentava (e ainda apresenta) uma biblioteca de componentes inteiramente desenvolvida em Object Pascal – a VCL (Visual Component Library) –, na qual cada componente era representado por uma classe. Além disso, a linguagem Object Pascal suportava os requisitos básicos de programação orientada a objetos (excetuando-se apenas os recursos de herança múltipla – que pode ser simulada – e sobrecarga de operadores e funções – esse último recurso já incorporado desde a versão 4 do produto).

Além disso, a Borland disponibilizou o código-fonte dos componentes da VCL, o que permitiu aos desenvolvedores compreender sua estrutura hierárquica e codificação, e também facilitou a expansão dessa biblioteca mediante a criação de novos componentes (desenvolvidos por terceiros).

Essa estratégia impulsionou o surgimento de um novo mercado – o mercado de desenvolvedores de componentes. Ao mesmo tempo em que incorpora ao Delphi os componentes padrões para criação de interface, a Borland permite que terceiros desenvolvam componentes capazes de realizar tarefas específicas.

Esse mercado se desenvolveu com tanta rapidez e eficiência que alguns desses componentes desenvolvidos por terceiros, como o Rave Reports (para a criação de relatórios), o TChart (para geração de gráficos), e o Intraweb (para desenvolvimento RAD para WEB) passaram a integrar o pacote oficial do Delphi.

Atualmente existe uma infinidade de componentes, e muitos são comercializados pela Internet. A maioria destes possui uma versão de avaliação, e muitos podem ser adquiridos com o código-fonte completo. Por terem sido desenvolvidos em Object Pascal, não precisam de nenhuma DLL ou arquivo adicional, e são facilmente integrados ao ambiente de desenvolvimento do Delphi (que também suporta componentes ActiveX, mas sem as vantagens dos componentes nativos – denominação dada aos componentes desenvolvidos em Object Pascal). Existem ainda os componentes Freeware, que podem ser utilizados sem custo algum.

Recomendo que você use e abuse dos componentes existentes no mercado, de forma a acelerar o processo de desenvolvimento dos seus aplicativos. Não se esqueça, no entanto, de adquirir legalmente os componentes que utilizar, pois, além dos aspectos legais envolvidos, a justa remuneração estimula os desenvolvedores de componentes a prosseguir no seu trabalho.

Infelizmente, a falta de respeito aos direitos autorais se manifesta em grande escala por todo o mundo, e o Brasil não é exceção. Devemos ter em mente, no entanto, que a prática da pirataria tende a desestimular os desenvolvedores de soluções, e no futuro você pode se tornar uma vítima do seu próprio crime, ao não encontrar – para uma futura versão do Delphi – um componente que possa simplificar o seu trabalho de desenvolvimento (componente este que havia sido desenvolvido para versões anteriores do produto). Esse tipo de argumento se aplica ao próprio Delphi, comercializado em diversas versões, a preços compatíveis para cada finalidade.

A fim de facilitar o desenvolvimento de aplicações multiplataforma, a Borland passou a incluir, desde a versão 6 do Delphi, uma nova biblioteca de componentes – denominada CLX – e usada pelo Kylix (a ferramenta RAD da Borland para o desenvolvimento de aplicações para o ambiente Linux). A CLX é baseada na biblioteca Qt da Troll Tech, que é realmente multiplataforma, e seus componentes são muito semelhantes àqueles existentes na VCL.

Conseqüentemente, o desenvolvedor que for capaz de, utilizando o Delphi 7 como ferramenta de desenvolvimento, criar aplicações para o ambiente Windows baseadas na CLX, não terá nenhuma dificuldade em migrar suas aplicações para o ambiente Linux, caso empregue o Kylix como ferramenta de desenvolvimento para aquele sistema operacional.

Um alerta aos iniciantes: programação não é sinônimo de criação de interface! Nos últimos anos, tenho observado que muitas pessoas (principalmente os iniciantes em programação) estão confundindo um pouco as coisas. Desenvolver um aplicativo requer mais do que apenas construir uma bela interface, e o objetivo das ferramentas RAD é exatamente esse: simplificar a criação da interface para permitir que o desenvolvedor se atenha mais à análise, projeto e codificação do sistema. A interface, como o próprio nome diz, serve para a comunicação entre o usuário e o sistema, e corresponde apenas a uma parcela do seu desenvolvimento.

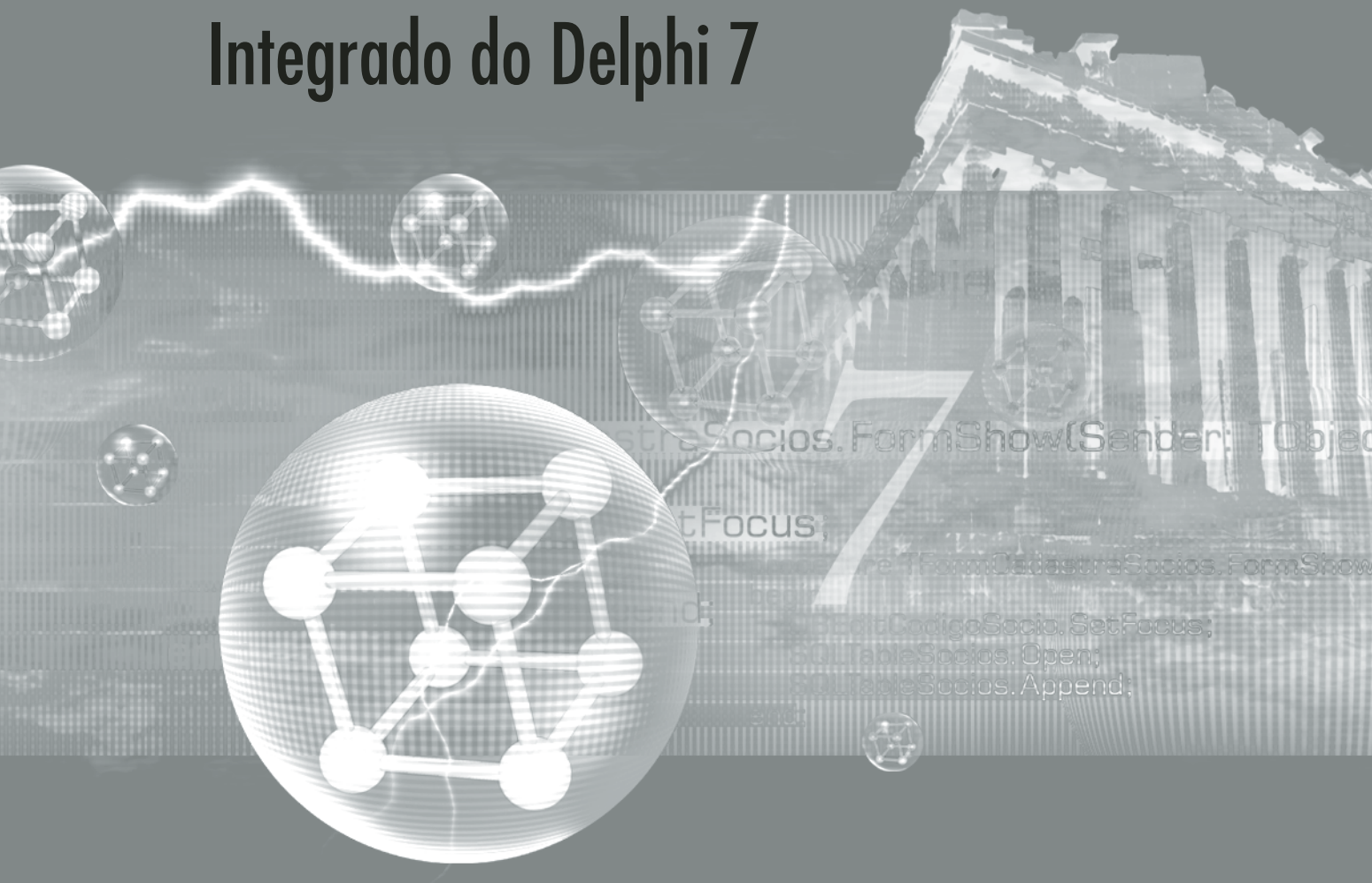
Desenvolver uma aplicação requer conhecimento de tópicos como, por exemplo, algoritmos, lógica de programação e teoria de Bancos de Dados. Devemos considerar que a formação de um desenvolvedor não se faz da noite para o dia, e a programação é apenas um dos ingredientes dessa formação.

Embora tenha me esforçado para redigir meus livros de forma didática e precisa, de maneira a poder oferecer informação atualizada aos nossos leitores, a rapidez com que a informática evolui força-nos a escrever e publicar livros em tempo recorde! Conseqüentemente, eventuais erros de redação ou digitação podem vir a ocorrer, mas que de forma alguma comprometem o entendimento do assunto. Reforço o argumento de que programação é uma atividade que exige muito raciocínio, estando distante de ser uma atividade mecânica. De qualquer maneira, estarei à disposição para resolver dúvidas referentes ao livro pelo endereço eletrônico mrleao@rio.com.br.

Gostaria de concluir esse capítulo parabenizando o leitor por adotar o Delphi 7 como ferramenta de desenvolvimento. Conforme será mostrado ao longo do livro, o Delphi 7 é ferramenta de desenvolvimento robusta e capaz de atender às necessidades de desenvolvedores de qualquer porte – desde aqueles que desenvolvem aplicativos mais simples (com acesso a bancos de dados locais) até aqueles que estão comprometidos com o desenvolvimento de soluções corporativas e para a Internet.

Capítulo 2

O Ambiente de Desenvolvimento Integrado do Delphi 7



Neste capítulo será apresentado o ambiente de desenvolvimento integrado do Delphi 7. Serão mostrados os conceitos fundamentais de formulários, controles, componentes, propriedades, métodos e eventos, cuja compreensão é indispensável ao desenvolvimento de uma boa aplicação.

Fundamentos em: Composição do Ambiente de Desenvolvimento

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente Windows.

METODOLOGIA

- Apresentação dos elementos que compõem o ambiente de desenvolvimento integrado do Delphi 7.

Apresentação do Ambiente de Desenvolvimento Integrado do Delphi 7

Neste capítulo será apresentado o ambiente de desenvolvimento do Delphi 7 e os conceitos fundamentais utilizados na criação da interface de um aplicativo.

As figuras que se seguem apresentam o ambiente de desenvolvimento integrado do Delphi 7, tal como se apresenta quando você o inicializa pela primeira vez.

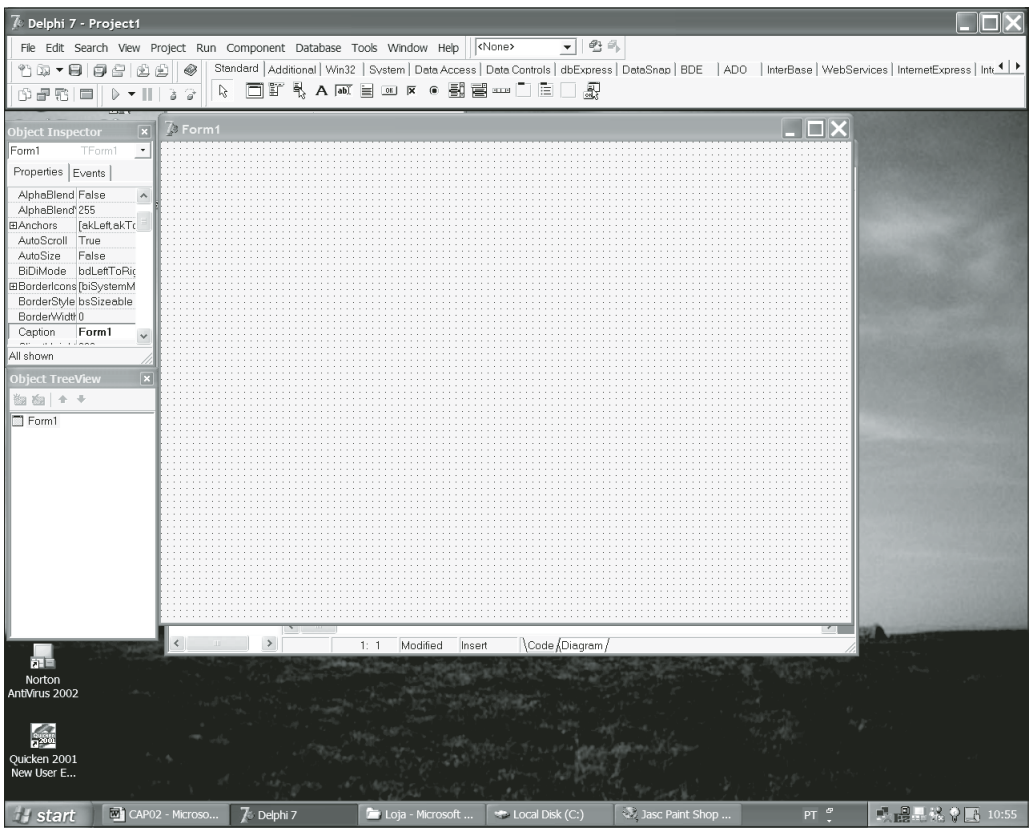


Figura 2.1: Ambiente de desenvolvimento integrado do Delphi 7.

As figuras exibidas neste livro foram capturadas com o sistema operacional Windows XP, mas os procedimentos e as janelas são muito semelhantes nas outras versões deste sistema operacional (95/98/Me/NT/2000).

O ambiente de desenvolvimento integrado do Delphi 7 é composto pelas seguintes áreas de trabalho:

a) Uma janela principal, onde se encontram:

- ♦ Uma barra de títulos, que exibe o título do projeto corrente. Quando você inicia um novo projeto, este recebe o nome default Project1, como mostra a figura anterior.
- ♦ Uma barra de menus, que dá acesso aos principais comandos e configurações do ambiente de desenvolvimento integrado do Delphi 7.
- ♦ Uma caixa de ferramentas, composta de vários botões reunidos em três grupos, e que fornecem atalho para a execução de diversas tarefas, como será mostrado ao longo do texto.
- ♦ Uma caixa combo que permite selecionar a configuração desejada para a sua área de trabalho.
- ♦ Uma paleta de componentes, composta por diversas páginas nas quais os componentes estão reunidos por categoria. Estes componentes são os objetos utilizados na criação da interface do programa, no desenvolvimento de aplicações para o ambiente Windows. Estas páginas da paleta de componentes apresentam algumas diferenças quando a aplicação se baseia na VCL ou na CLX. A VCL, por ser desenvolvida exclusivamente para o ambiente Windows, possui mais componentes que a CLX (que precisa atender ao requisito de ser multiplataforma). Logo, se você for desenvolver uma aplicação exclusivamente para o ambiente Windows, deverá dar preferência à utilização da VCL. Caso a aplicação venha a ser multiplataforma (compilada no Delphi para o ambiente Windows e no Kylix para o ambiente Linux), deverá dar preferência à utilização da CLX.

b) Uma janela chamada Object Inspector, que dá acesso às principais propriedades e eventos de um componente. Esta janela é composta por:

- ♦ Uma caixa de seleção de objetos, utilizada para selecionar o objeto cujas propriedades e eventos desejamos alterar.
- ♦ Duas guias, intituladas Properties e Events, que dão acesso às páginas de propriedades e de eventos, respectivamente.

Os nomes das propriedades e dos eventos podem ser organizados em ordem alfabética ou por categoria, além de se poder ocultar a exibição de determinados tipos de propriedades e eventos. Esta versatilidade permite otimizar ainda mais a utilização do ambiente de desenvolvimento.

c) Uma janela chamada Form1, criada automaticamente pelo Delphi 7. O que o Delphi 7 chama de formulários são as janelas usadas no desenvolvimento de aplicações. Em geral, uma aplicação é composta por diversas janelas (ou, como descrito anteriormente, por vários formulários).

d) Uma janela denominada Code Editor – mostrada na figura a seguir –, inicialmente sobreposta pelo formulário Form1, na qual será digitado o código da aplicação. Para alternar entre o Editor de Códigos e o formulário, basta pressionar seguidamente a tecla de função F12.

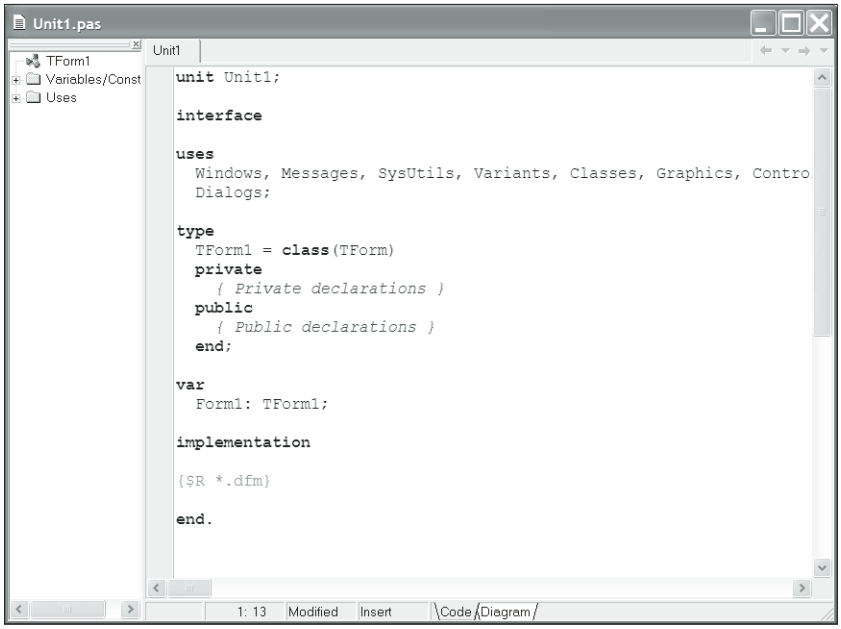


Figura 2.2: O Editor de Códigos do Delphi 7.

Para que uma aplicação se comporte da maneira desejada, torna-se necessária a inclusão de linhas de código que definam o comportamento do programa, como resposta a ações do usuário ou do sistema operacional.

Embora o Delphi 7 facilite muito o trabalho de codificação de um aplicativo, sempre será necessário incluir algumas linhas de código, como será mostrado ao longo do livro, no desenvolvimento dos nossos aplicativos-exemplo.

Repare que, inicialmente, a janela do Editor de Códigos possui uma outra janela "ancorada" ao longo da sua extremidade esquerda. Esta janela, chamada Code Explorer, e que pode ser reposicionada dentro do ambiente de desenvolvimento integrado do Delphi 7, permite uma melhor movimentação pelo código da aplicação, principalmente quando os arquivos de código se tornam mais extensos.

Em sua barra de títulos, a janela do Code Editor apresenta o nome do arquivo em que está armazenado o código que está sendo editado no momento. Inicialmente, conforme mostrado na figura anterior, esse arquivo é denominado Unit1.

Logo abaixo da sua barra de títulos são exibidas as guias correspondentes aos arquivos de código que estão abertos. Na figura anterior, como apenas o arquivo unit1 está aberto, só existe uma guia, identificada pelo nome Unit1 (essa é a situação existente quando se inicia um novo projeto).

Ao editar o código do seu programa, você pode aplicar as técnicas de copiar, recortar e colar texto usando as combinações de tecla Ctrl+C, Ctrl+X e Ctrl+V, como se faz em muitos editores de texto, ou empregar os itens correspondentes no menu Edit da janela principal do ambiente de desenvolvimento.

e) Uma janela chamada Object TreeView – mostrada na figura a seguir – que permite selecionar rapidamente um componente inserido no formulário, estando os nomes dos componentes exibidos

visualmente em uma estrutura hierárquica. Esta janela é muito útil quando se deseja alterar as propriedades de um componente cuja seleção pode ser dificultada pela existência de outros componentes sobrepostos e que ocupem boa parte da sua região visível.

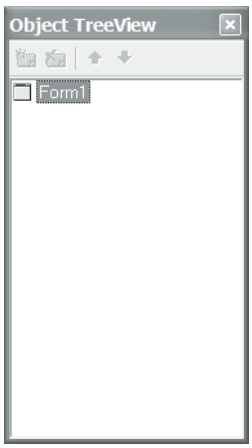


Figura 2.3: A Janela Object TreeView.

É importante destacar que o ambiente de desenvolvimento do Delphi 7 é completamente configurável. Você pode ancorar janelas em outras, além de arrastá-las e reposicioná-las de forma que o ambiente seja configurado para acelerar ao máximo a sua produtividade.

Existem ainda outras janelas, usadas principalmente na depuração de aplicativos, e que serão discutidas em um capítulo posterior.

A janela principal é usada para a edição do trabalho, personalização do ambiente, gerenciamento de documentos, etc. Os elementos que compõem a janela principal podem ser tratados como barras de ferramentas independentes (mas internas à janela principal), isto é, podem ser arrastados, reposicionados, etc. dentro da janela principal.

Você pode selecionar os elementos que devem ser exibidos na janela principal. Para isso, basta executar o seguinte procedimento:

1. Pressionar o botão direito do mouse sobre qualquer área livre da janela principal (geralmente na área superior direita dessa janela, próximo ao menu Help) para exibir o menu pop-up mostrado na figura a seguir, na qual se pode selecionar os elementos da janela principal que devem ou não ser exibidos.

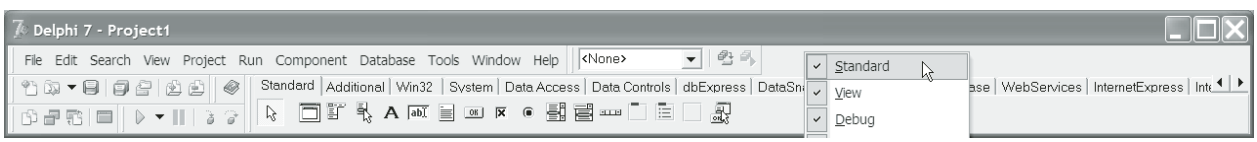


Figura 2.4: Exibindo o menu pop-up da janela principal do ambiente de desenvolvimento do Delphi 7.

Você também pode selecionar o item Customize desse menu, para exibir a caixa de diálogo Customize, mostrada na figura a seguir.

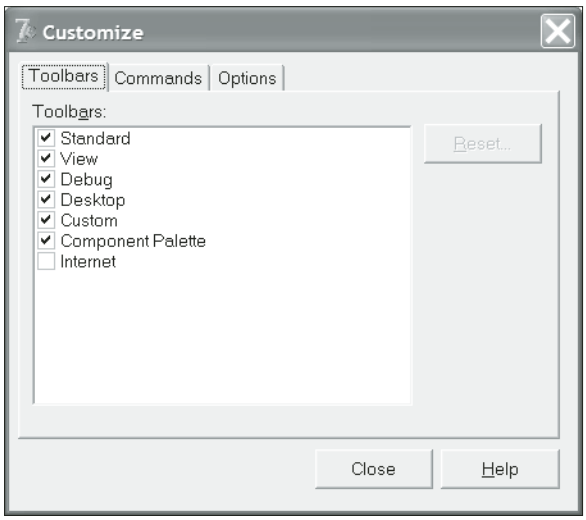


Figura 2.5: A caixa de diálogo Customize.

Essa caixa de diálogo apresenta ainda as guias Commands e Options, que permitem que se definam opções para a janela principal.

Utilizando-se da guia Commands dessa caixa de diálogo, você pode adicionar novos botões às barras de ferramentas da janela principal. Para isso, basta executar os seguintes procedimentos:

1. Exiba a caixa de diálogo Customize, executando os procedimentos descritos anteriormente.
2. Selecione a guia Commands dessa caixa de diálogo.
3. Selecione uma das categorias na lista da esquerda (lista Categories). Os comandos correspondentes são exibidos na lista da direita (Commands).
4. Selecione o comando que deseja acessar por meio de um botão de comando e o arraste para a janela principal, na posição em que o botão deve ser inserido. Caso o comando possua um bitmap associado, este será exibido no botão a ser criado; caso contrário, o texto do comando será exibido (e as dimensões do botão serão maiores).

Para remover o botão de comando, basta selecioná-lo e arrastá-lo para fora da barra de ferramentas.

Utilizando a guia Options dessa caixa de diálogo, você pode definir se os botões de comando devem ou não exibir strings de auxílio e se estas devem ou não exibir as teclas de atalho correspondentes ao comando. Para isso, basta selecionar as opções correspondentes.

Formulários – Os Elementos de Criação da Interface com o Usuário

Quando você inicia o ambiente de trabalho do Delphi 7 ou começa a desenvolver uma nova aplicação, aparece um formulário default denominado Form1, criado automaticamente pelo ambiente, no qual você pode começar a construir a interface visual do seu programa.

O que o Delphi 7 chama de formulário é, na realidade, uma janela de uma aplicação, que pode ser a sua janela principal, uma caixa de diálogo, etc.

Um formulário apresenta uma área de trabalho, denominada área-cliente, na qual podem ser incluídos os controles e componentes que proporcionarão maior funcionalidade à sua aplicação.

A figura a seguir mostra o formulário default criado pelo Delphi 7.

Figura 2.6: O formulário default criado pelo Delphi 7.

Os procedimentos necessários à personalização de um formulário serão apresentados posteriormente, à medida que forem apresentados os procedimentos necessários à criação da interface.

Controles e Componentes

Os componentes são os objetos utilizados para criar a interface do programa com o usuário e executar tarefas específicas do sistema operacional.

Os controles são os componentes que podem permanecer visíveis durante a execução do aplicativo. Fica evidente, portanto, que os componentes podem ser divididos em dois grandes grupos: os componentes não-visuais e os componentes visuais – os controles.

Como exemplo de componentes não-visuais podem ser citados os que permitem a exibição das caixas de diálogo padrão do sistema (encontradas na página Dialogs da paleta de componentes) e o temporizador (encontrado na página System da paleta de componentes do Delphi 7), usado para gerar mensagens do sistema a intervalos de tempo predefinidos.

Como exemplo de controles podemos citar as caixas de texto, rótulos, caixas de lista e botões de comando, entre muitos outros que serão mostrados ao longo do livro.

Em suma, os controles são os componentes visuais com os quais o usuário poderá interagir durante a execução do aplicativo. Os componentes não-visuais, por outro lado, permanecem invisíveis durante a execução da aplicação (o usuário não os enxerga), embora realizem tarefas importantes.

Os controles e componentes estão distribuídos, de acordo com a sua funcionalidade, pelas diversas páginas da paleta de componentes (Figura 2.7), situada na janela principal do Delphi 7.

Figura 2.7: A paleta de componentes do Delphi 7.

As páginas da paleta de componentes exibidas no Delphi 7 dependem da biblioteca de componentes utilizada. As páginas de componentes da VCL e da CLX não são exatamente iguais, mas a maneira de utilizá-las é a mesma. As diferenças dizem respeito à existência ou não de determinadas páginas (A CLX, por ser multiplataforma, não possui uma página chamada Win32 na sua paleta de componentes) e componentes específicos, bem como à localização de determinados componentes. O temporizador (componente Timer) é um exemplo de componente disponível nas duas bibliotecas, mas em páginas diferentes da paleta de componentes.

É importante destacar que, inicialmente, nem todas as guias estão visíveis. Para navegar pelas guias da paleta de componentes, use os botões de seta situados no seu canto superior direito. Para selecionar uma das páginas, basta clicar com o botão esquerdo do mouse sobre a guia correspondente.

Quando o número de componentes de uma página exceder a sua capacidade de exibição, use os botões de seta situados nas extremidades da página para exibir os componentes que estiverem ocultos. Outra opção é alterar a resolução atual do seu monitor, de forma a tornar possível a exibição simultânea de um maior número de componentes por página. Normalmente, em uma resolução de 800 x 600, todos os componentes podem ser acessados sem qualquer dificuldade, o que não ocorre no caso em que se adota uma resolução de 640 x 480.

Cada controle ou componente tem propriedades, métodos e eventos associados. As propriedades de um controle ou componente definem o seu aspecto e algumas das suas principais características. Os métodos são funções ou procedimentos intrínsecos ao controle ou componente, e são capazes de realizar alguma tarefa específica quando executados. Os eventos, por sua vez, são acontecimentos associados a um controle ou componente. As definições de propriedades, métodos e eventos serão detalhadas nos tópicos a seguir.

Propriedades

Os controles e componentes, junto com os formulários, são objetos predefinidos da linguagem Object Pascal (a linguagem de programação utilizada pelo Delphi 7) e apresentam algumas características muito semelhantes às dos objetos que diariamente manipulamos. Vamos aproveitar essas semelhanças na definição das características dos objetos da linguagem Object Pascal e eleger como objeto de

comparação um monitor de vídeo, objeto que todos nós, programadores, conhecemos e com o qual estamos habituados a trabalhar.

Todo e qualquer objeto tem propriedades que o caracterizam e o diferenciam dos demais. Entre as propriedades de um monitor de vídeo, podemos destacar:

- O seu tamanho (definido em polegadas).
- A cor do seu gabinete.

Assim como o monitor de vídeo, um formulário (e todos os demais objetos da linguagem) também tem propriedades que o diferenciam, tais como:

- A sua cor, que é definida pelo valor armazenado na sua propriedade Color.
- O seu tamanho, definido pelo valor armazenado nas suas propriedades Height (altura) e Width (largura).

A definição ou alteração de um valor para uma propriedade pode ser feita de duas formas:

- Na fase de projeto, usando-se o Object Inspector (que será visto em um tópico posterior, ainda neste capítulo).
- Durante a execução do aplicativo, mediante a inclusão de uma linha de código com a seguinte sintaxe:

```
nome_do_objeto.nome_da_propriedade:= valor;
```

Esse é um típico exemplo de comando de atribuição, e o sinal de igualdade precedido por dois-pontos é, nesse caso, denominado operador de atribuição. Os operadores da linguagem Object Pascal serão apresentados no próximo capítulo.

Métodos

Todo objeto na vida real tem alguma funcionalidade e, no caso do monitor de vídeo, por exemplo, essa funcionalidade consiste em mostrar a imagem correspondente a um sinal enviado pela placa de vídeo do sistema. Entretanto, para que isso aconteça, é necessário que o monitor de vídeo saiba como transformar o sinal emitido pela placa de vídeo em imagem, isto é, deve existir um *método* para se fazer isso (no caso do monitor de vídeo, existem circuitos eletrônicos que se encarregam dessa tarefa).

Assim como o monitor de vídeo, os objetos utilizados no desenvolvimento de uma aplicação com o Delphi 7 também terão métodos que os tornarão capazes de realizar determinadas tarefas. Diferentemente do que ocorre com as propriedades, que também podem ser definidas na fase de projeto, a chamada a um método só pode ser feita durante a execução do aplicativo, mediante a inclusão de uma linha de código com a seguinte sintaxe:

```
nome_do_objeto.nome_do_método;
```

Conforme será descrito posteriormente (no capítulo referente à programação orientada a objetos), um método pode ou não receber parâmetros, sendo, na realidade, uma função ou procedimento que é definida para o objeto (para ser mais exato, um método é definido internamente a uma classe, da qual

o objeto é uma instância, mas por enquanto vamos deixar de lado esses preciosismos de definição, de forma a não confundir o prezado leitor).

Em alguns casos, pode ser necessária a passagem de parâmetros na chamada a um método.

Eventos

No tópico anterior vimos que todo objeto tem métodos, que são características que representam a sua funcionalidade.

Normalmente, os métodos são executados em resposta a algum acontecimento (um *evento*). No caso do monitor de vídeo, por exemplo, quando o usuário alterna entre aplicações do Windows (quando ocorre o *evento* de alternar entre aplicações), os métodos necessários à atualização da imagem são executados.

Da mesma forma que no monitor de vídeo, os objetos que usamos no desenvolvimento dos nossos aplicativos com o Delphi 7 também apresentam eventos associados.

Cada objeto tem a capacidade de responder a um determinado conjunto de eventos, e essa resposta a um evento é feita mediante a definição de um procedimento associado a esse evento, e esse procedimento pode ser facilmente acessado por meio do Object Inspector.

Sempre que um determinado evento ocorre, a aplicação verifica se existe um procedimento associado a esse evento e, em caso positivo, o procedimento a ele associado é executado.

Considere, por exemplo, um botão de comando, objeto bastante comum nas aplicações desenvolvidas para o ambiente Windows. Quando um usuário seleciona um botão (clicando sobre este com o botão esquerdo do mouse), diz-se que ocorreu o evento de "clicar sobre o botão com o botão esquerdo do mouse". Quando esse evento ocorre, o usuário espera alguma resposta (afinal de contas, se o botão está lá, deve servir para alguma coisa). Associado a esse evento, deve haver um procedimento a ser executado sempre que isso ocorrer. É importante lembrar que, ao ocorrer esse evento, a aplicação verifica se existe um procedimento associado. Se houver, esse procedimento é executado, se não, a aplicação não faz nada. Cabe a você – o desenvolvedor – definir um procedimento associado a esse evento e codificar os comandos a serem executados nesse procedimento.

Nos próximos tópicos, serão apresentados os passos necessários à definição de procedimentos associados a eventos.

O Object Inspector

Essa janela, que fornece acesso direto às propriedades e eventos associados a um componente, tem:

- Uma caixa de seleção de objetos, que identifica o objeto selecionado.
- Páginas de eventos (acessada selecionando-se a guia Events com o botão esquerdo do mouse) e de propriedades (acessada selecionando-se a guia Properties com o botão esquerdo do mouse).

A Figura 2.8 mostra a página de propriedades do objeto Form1.

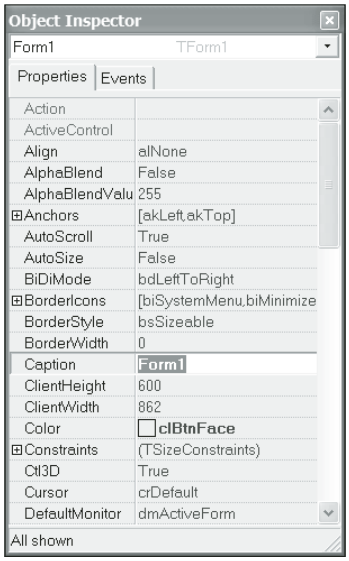


Figura 2.8: A página Properties do Object Inspector no Delphi 7.

Alterando o Valor de uma Propriedade no Object Inspector

Você pode alterar o valor de uma propriedade de um componente executando os seguintes procedimentos:

1. Selecione o componente, clicando sobre ele com o botão esquerdo do mouse ou por meio da caixa de seleção de objetos do Object Inspector.
2. Selecione a página Properties do Object Inspector, se ela já não estiver selecionada.
3. Clique no campo à direita da propriedade a ser alterada.
4. Defina o novo valor da propriedade.

Quando se altera a propriedade de um componente usando-se o Object Inspector, diz-se que essa propriedade está sendo alterada na fase de projeto do aplicativo. Quando essa alteração for feita mediante a inclusão de uma linha de código (conforme descrito anteriormente), diz-se que a propriedade está sendo alterada durante a execução do aplicativo.

Conforme descrito anteriormente, você pode optar por exibir as propriedades em ordem alfabética ou agrupadas por categoria, bastando selecionar a opção correspondente no item Arrange do menu pop-up que é exibido quando você pressiona o botão direito do mouse sobre o Object Inspector, como mostra a Figura 2.9.

Você também pode selecionar as propriedades que devem estar visíveis, bastando marcar ou desmarcar a opção correspondente no item View do menu pop-up apresentado na figura anterior. Além disso, o Object Inspector exibe, em sua extremidade inferior, o número de propriedades ou eventos ocultos (ou a frase All Shown – indicando que todos estão sendo exibidos).

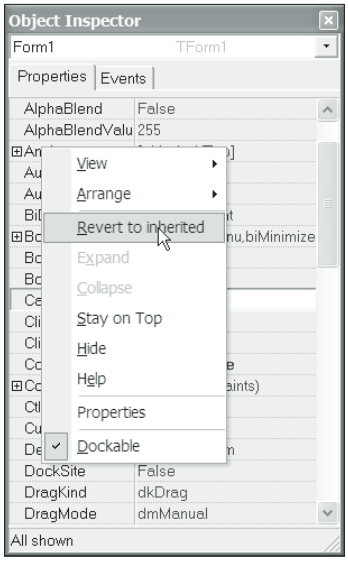


Figura 2.9: O menu pop-up do Object Inspector.

Definindo Procedimentos Associados a Eventos

A Figura 2.10 mostra a página Events do object Inspector, usada para definir procedimentos associados a eventos para um objeto.

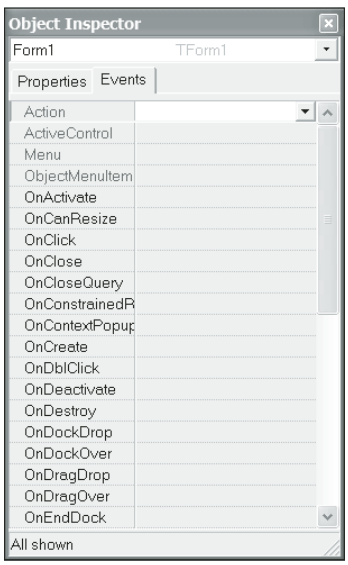


Figura 2.10: A página Events do Object Inspector.

Para alguns componentes, algumas propriedades consideradas muito importantes também são exibidas na página de eventos do Object Inspector.

Você pode definir um procedimento associado a um evento de um objeto da seguinte forma:

1. Selecione o objeto clicando sobre ele com o botão esquerdo do mouse ou por meio da caixa de seleção de objetos.
2. Selecione a página Events do Object Inspector, se ela já não estiver selecionada.
3. Dê um duplo clique no espaço em branco exibido à direita do nome do evento cujo procedimento deverá ser definido. Será exibida a janela do Code Editor, com o cursor situado no ponto em que deve ser inserido o código associado àquele evento. A Figura 2.11 mostra a janela exibida quando se cria o procedimento associado ao evento OnActivate do objeto Form1.
4. Digite o trecho de código a ser executado quando o evento ocorrer.

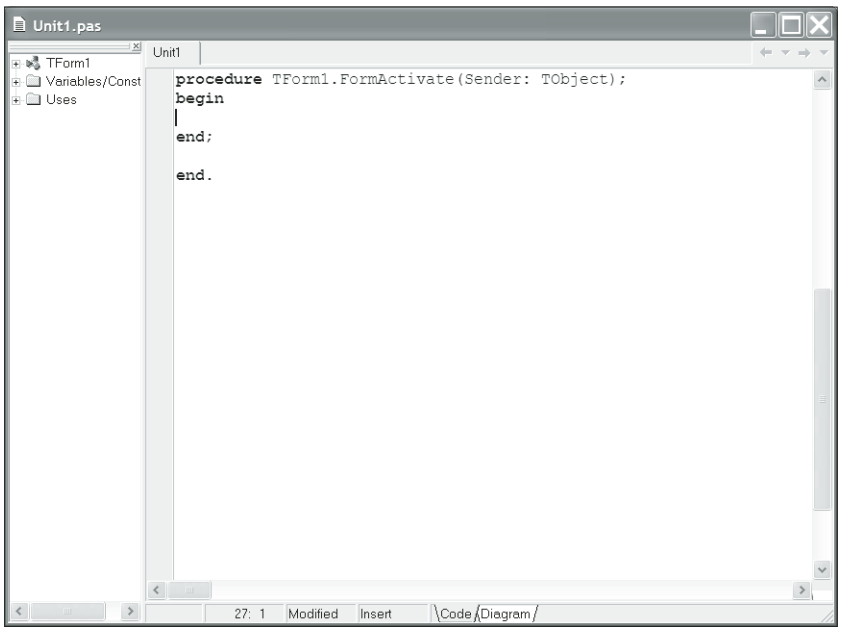


Figura 2.11: Definindo o procedimento associado a um evento.

Observe que o cabeçalho e o corpo principal do procedimento são gerados automaticamente. Qualquer código a ser executado em resposta a esse evento deverá ser escrito entre as palavras *begin* e *end* (não as apague). Como mostra a Figura 2.12, além de criar o cabeçalho e o corpo principal da função, o compilador ainda inseriu uma declaração do procedimento dentro da definição da classe TForm1 (Calma! Se você não sabe o que é uma classe, não precisa se assustar, pois isso será visto no próximo capítulo).

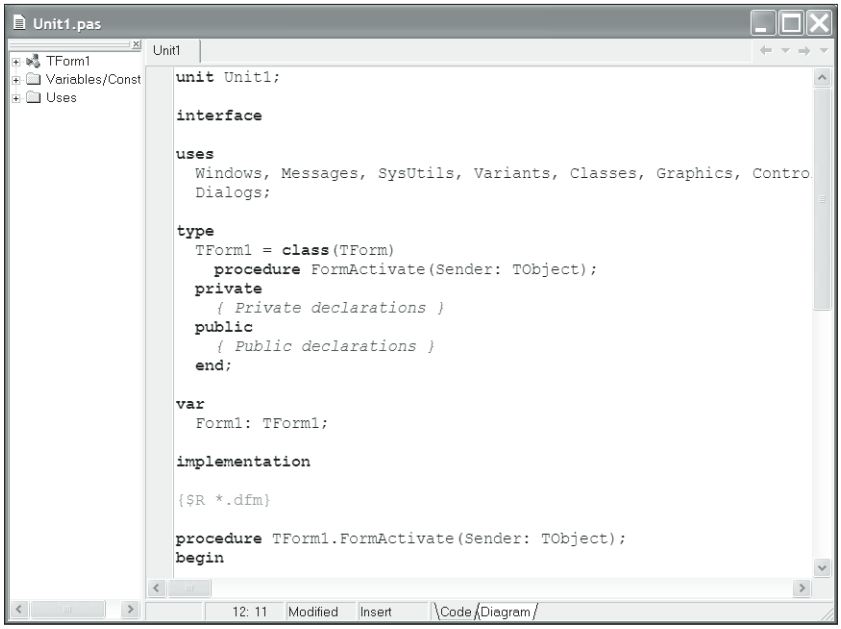


Figura 2.12: Declaração do procedimento associado a um evento.

O Conceito de Projeto de uma Aplicação

Normalmente, uma aplicação desenvolvida para o ambiente Windows tem diversas janelas e caixas de diálogo, cada uma com sua finalidade específica (entrada de dados, mensagens de advertência, exibição de gráficos, etc.) e diversos arquivos de código.

Dessa maneira, fica claro que o desenvolvimento de uma aplicação com o Delphi 7 exigirá a utilização de diversos formulários e arquivos de código.

Como a quantidade de formulários e arquivos de códigos de uma aplicação pode tornar difícil o seu gerenciamento por parte do programador, o Delphi 7 utiliza o conceito de Projeto de uma aplicação. Quando você inicia o desenvolvimento de uma aplicação, está, na realidade, criando um projeto. Quando você adiciona um formulário ou arquivo de código para a aplicação, está, na realidade, adicionando arquivo(s) ao projeto. O Projeto chama para si a responsabilidade de gerenciar a aplicação, facilitando o trabalho do programador.

Conforme descrito anteriormente, quando você inicia um novo projeto de aplicação no Delphi 7, ele recebe o nome de Project1 (o nome do projeto é exibido na barra de título da janela principal) e será armazenado no arquivo project1.dpr, a menos que você o renomeie, selecionando o item Save Project As do menu File e redefinindo os nomes dos arquivos de código e de projeto nas caixas de diálogo correspondentes.

Além do arquivo de projeto, o Delphi 7 cria também um formulário (denominado Form1) e uma unidade de código, denominada unit1 e armazenada no arquivo unit1.pas (a menos que você o renomeie, selecionando o item Save As do menu File ou definindo esse nome na caixa de diálogo Save Unit1 As, exibida ao se salvar o projeto).

Um projeto de aplicação tem a sua definição armazenada em um arquivo com a extensão DPR – e pode ser visualizado selecionando-se o item View Source do menu Project. A Figura 2.13 mostra o código do arquivo project1.dpr, criado automaticamente pelo Delphi 7.

Observe que agora existem dois arquivos de código abertos no Code Editor (sendo que um deles é o arquivo de projeto). Você pode exibir um arquivo selecionando a guia correspondente.

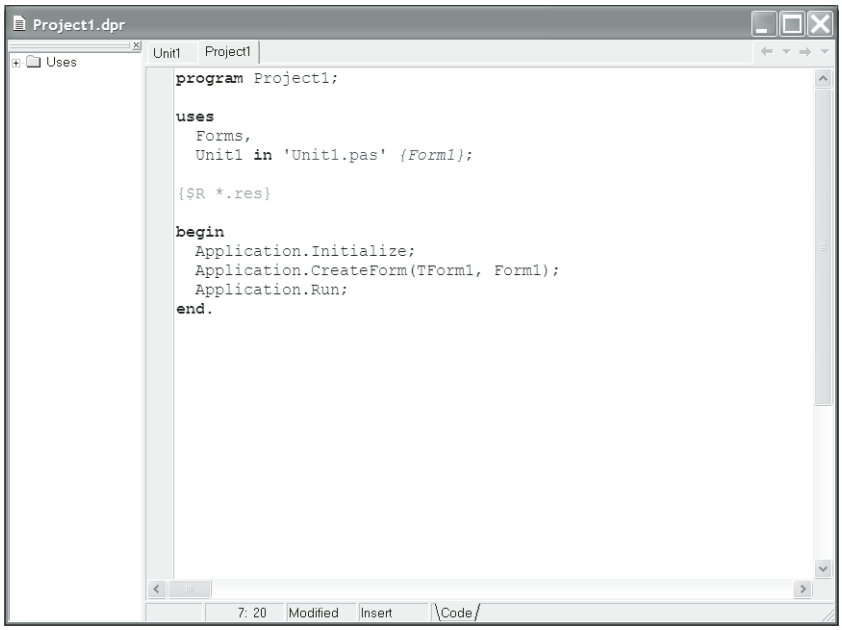


Figura 2.13: O código do arquivo de projeto.

Na primeira linha, tem-se a palavra-chave *program* seguida do nome do projeto (nome do arquivo de projeto). Se você salvar esse projeto com um outro nome, esse outro nome será mostrado logo após a palavra-chave *program* (em vez de Project1).

O programador iniciante não deve manipular diretamente o arquivo de projeto. Mesmo no desenvolvimento de aplicativos avançados, raramente se faz necessária a manipulação direta desse arquivo, ficando todo o gerenciamento do projeto por conta do próprio ambiente.

Em seguida pode-se verificar a cláusula Uses reproduzida a seguir.

Na VCL:

```
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
```

Na CLX

```
uses
  QForms,
  Unit1 in 'Unit1.pas' {Form1};
```

Essa cláusula permite que o arquivo de projeto acesse as definições armazenadas nos arquivos Forms.pas (no caso da VCL) ou QForms.pas (no caso da CLX) e Unit1.pas.

Repare que, enquanto a VCL usa a unit Forms, a CLX usa a unit QForms. Em geral, o prefixo Q identifica que a unit pertence à biblioteca CLX.

Em seguida verifica-se a presença da seguinte linha de código:

```
{$R *.res}
```

Esta linha de código, exibida normalmente em itálico, representa uma diretiva de compilação, e indica ao compilador que, associado a este projeto, deve existir um arquivo de recursos (resources) com o mesmo nome do arquivo de projeto e extensão .res, e que se este arquivo não existir, deverá ser criado.

Por fim, verifica-se a existência do seguinte trecho de código (que é o código principal do arquivo de projeto, responsável pela execução do aplicativo):

```
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Nesse trecho de código usa-se o objeto Application (que é uma instância da classe TApplication) que representa a aplicação.

O método Initialize do objeto Application, executado na linha de código reproduzida a seguir, é responsável pela inicialização da aplicação.

```
Application.Initialize;
```

O método CreateForm do objeto Application carrega um formulário na memória (esse método recebe como parâmetros a classe e o nome do formulário). Nesse caso, como o projeto recém-criado já possui automaticamente um formulário chamado Form1, da classe TForm1, a criação desse formulário é feita na execução da seguinte linha de código:

```
Application.CreateForm(TForm1, Form1);
```

A execução da aplicação se inicia com a execução do método Run do objeto Application, o que ocorre mediante a execução da seguinte linha de código:

```
Application.Run;
```

Se você examinar o código do arquivo unit1.pas criado pela VCL, mostrado a seguir, verá que esse código faz referência ao objeto Form1:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Na primeira linha, tem-se a palavra-chave *unit* seguida do nome da unit (nome da unidade de código). Se você salvar o arquivo dessa unit com um outro nome, esse outro nome será mostrado logo após a palavra-chave *unit* (em vez de unit1).

Além disso, pode-se observar a seguinte linha de código:

```
{$R *.DFM}
```

Essa linha de código representa uma diretiva de compilação e indica ao Delphi 7 que, associado a essa unit, existe um arquivo de mesmo nome, mas com a extensão DFM.

Um arquivo com a extensão DFM armazena uma descrição textual do formulário.

Para acessar o arquivo unit1.dfm, por exemplo, basta executar os seguintes procedimentos:

1. Selecione o formulário, clicando sobre este com o botão esquerdo do mouse.
2. Pressione o botão direito do mouse, para exibir o menu pop-up mostrado na Figura 2.14.

Nunca altere o nome do arquivo no qual a unit está armazenada usando as ferramentas disponíveis no sistema operacional. Sempre que precisar alterar o nome do arquivo no qual uma unit está armazenada, use as opções disponíveis no ambiente de desenvolvimento (usando, por exemplo, o item Save as do menu File ou o botão correspondente da barra de ferramentas).

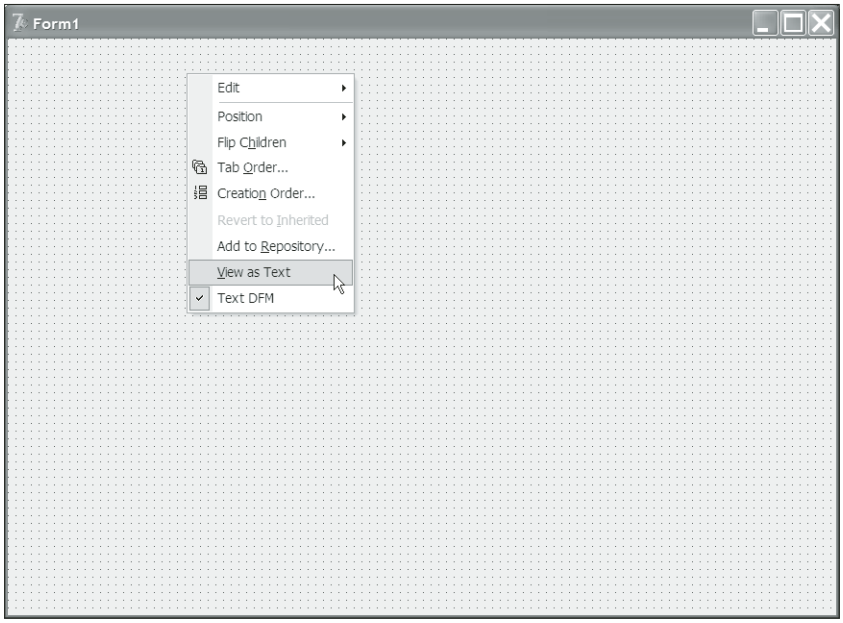


Figura 2.14: Exibindo o menu pop-up de um formulário.

3. Selecione o item View as Text do menu pop-up. Pronto! Será exibida a descrição textual do formulário, como mostra a Figura 2.15.

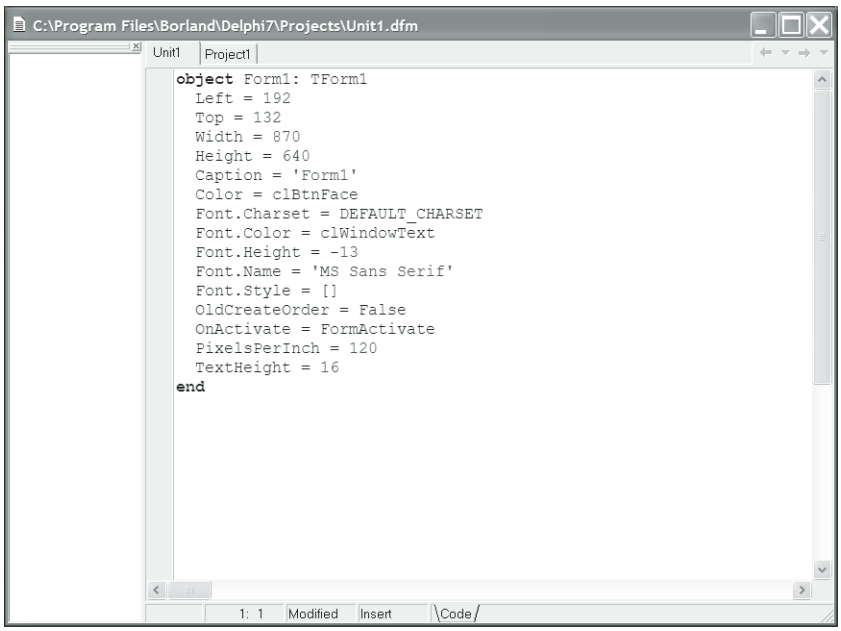


Figura 2.15: Visualizando a descrição textual de um formulário.

Até a versão 4 do Delphi esse arquivo, que contém a descrição textual do formulário, era armazenado com uma formatação especial e não podia ser visualizado em outros editores de texto. Nas últimas versões, este arquivo passou a ser armazenado, por default, como um arquivo ASCII, desde que a opção correspondente (Text DFM) esteja selecionada no menu pop-up do formulário.

No caso da CLX, o arquivo Unit1.pas apresenta a seguinte definição:

```
unit Unit1;

interface

uses
  SysUtils, Types, Classes, Variants, QTypes, QGraphics, QControls, QForms,
  QDialogs, QStdCtrls;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.xfm}

end.
```

Repare na diferença entre os nomes das units básicas acessadas pela VCL e pela CLX. Além disso, a diretiva de compilação {$R *.xfm} indica que, associado a essa unit, existe um arquivo de mesmo nome, mas com a extensão xfm (mantenha a extensão em letras minúsculas, pois caso você venha a recompilar sua aplicação no Kylix, isto faz diferença).

Podemos então concluir que, em qualquer das bibliotecas de componentes, todo formulário tem uma unidade de código associada (um arquivo com a extensão PAS) e possui a sua descrição visual armazenada em um arquivo com a extensão DFM (no caso da VCL) ou xfm (no caso da CLX) – que armazena a sua descrição textual.

Para retornar à situação anterior (representação visual do formulário), bastando executar os seguintes procedimentos:

1. Selecione o arquivo com a descrição textual do formulário.
2. Pressione o botão direito do mouse, para exibir o menu pop-up mostrado na Figura 2.16.
3. Selecione o item View As Form do menu pop-up. Pronto! Será exibida novamente a representação visual do formulário.

Ao fazer o backup de um projeto de aplicação, você deverá copiar:

- Os arquivos de código (extensão .pas).
- Os arquivos de descrição textual de formulário (extensão .dfm – no caso da VCL ou .xfm – no caso da CLX).
- O arquivo de projeto (extensão .dpr).
- Além dos arquivos do banco de dados, se for o caso.

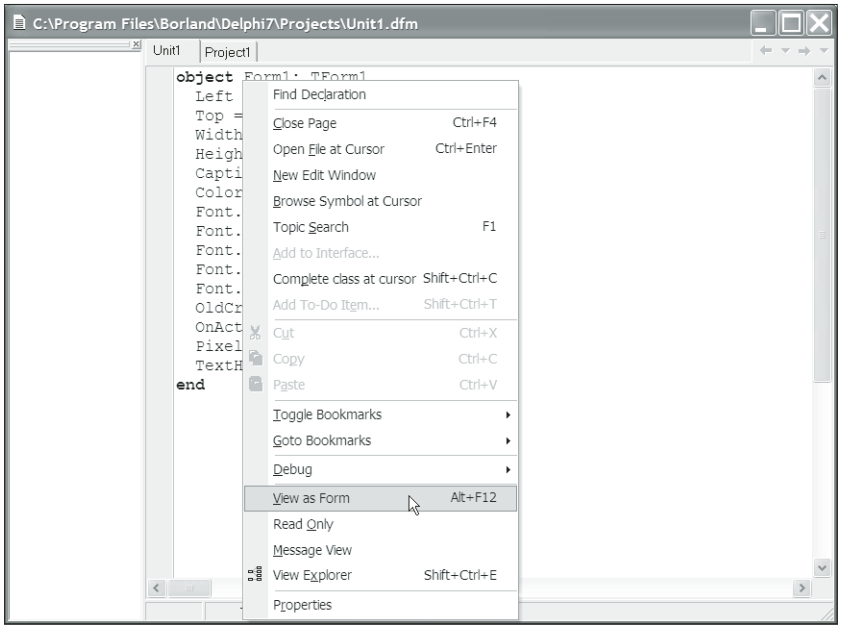


Figura 2.16: Exibindo o menu pop-up de um arquivo de formulário.

Iniciando um Novo Projeto

Sempre que você for iniciar o desenvolvimento de uma nova aplicação usando o Delphi 7 como ferramenta de desenvolvimento, deverá iniciar um novo projeto.

É recomendável que o seu projeto e todos os seus arquivos sejam salvos em uma pasta própria.

Nesta parte do livro, será desenvolvida uma aplicação para cadastrar os sócios de um clube, e suas respectivas atividades.

Para salvar nosso projeto e seus arquivos, será criada uma pasta denominada Clube.

Dessa maneira, devemos iniciar um novo projeto, referente a essa nova aplicação.

Para iniciar um novo projeto, execute um dos seguintes procedimentos:

1. Selecione o item New/Application do menu File, para criar uma aplicação baseada na VCL.

Ou:

1. Selecione o item New/CLX Application do menu File, para criar uma aplicação baseada na CLX.

Ou:

1. Selecione o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items (Figura 2.17).

Ou:

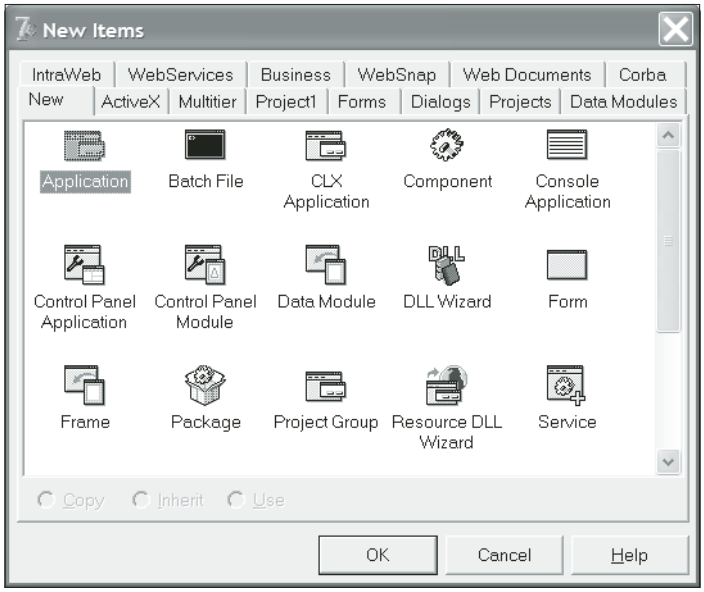


Figura 2.17: A caixa de diálogo New Items do Delphi 7.

1. Selecione o item Application, na página New desta caixa de diálogo, para criar uma aplicação baseada na VCL.
2. Selecione o item CLX Application, na página New desta caixa de diálogo, para criar uma aplicação baseada na CLX.
3. Clique em OK, para fechar a caixa de diálogo.

Será então criado um novo projeto chamado Project1, armazenado no arquivo Project1.dpr; um formulário denominado Form1, armazenado no arquivo unit1.dfm (no caso da VCL) ou unit1.xfm (no caso da CLX) e um arquivo de código associado ao formulário Form1, denominado Unit1.pas. Esses são os nomes default fornecidos pelo ambiente de desenvolvimento, mas você deve salvar o seu projeto e os seus arquivos de código com nomes mais fáceis de se memorizar.

Antes de iniciar o desenvolvimento de uma aplicação, você deve planejá-la com cuidado, principalmente no que se refere aos nomes dos arquivos a serem manipulados pela aplicação. Um arquivo com o código referente ao cadastro de um sócio, por exemplo, pode ser armazenado com o nome UnitCadastroSocio.pas. Repare que o nome de um arquivo deve ser o mais descritivo possível, de maneira a facilitar uma futura manutenção do sistema. Se após iniciar o desenvolvimento da aplicação você desejar alterar o nome de um arquivo de código após referenciá-lo em outros arquivos, será necessário alterar explicitamente todas as referências a este arquivo.

Salvando o Projeto Recém-criado

Para salvar um projeto recém-criado, execute um dos seguintes procedimentos:

1. Selecione o item Save Project As do menu File.

Ou:

1. Na caixa de ferramentas, selecione o ícone correspondente.

Inicialmente será exibida a caixa de diálogo Save Unit1 As (Figura 2.18), para que sejam definidos o nome e a pasta da primeira unidade de código do programa (que o ambiente chamou de Unit1). Salve essa unidade de código com um nome fácil de se memorizar, como UnitPrincipal, por exemplo.

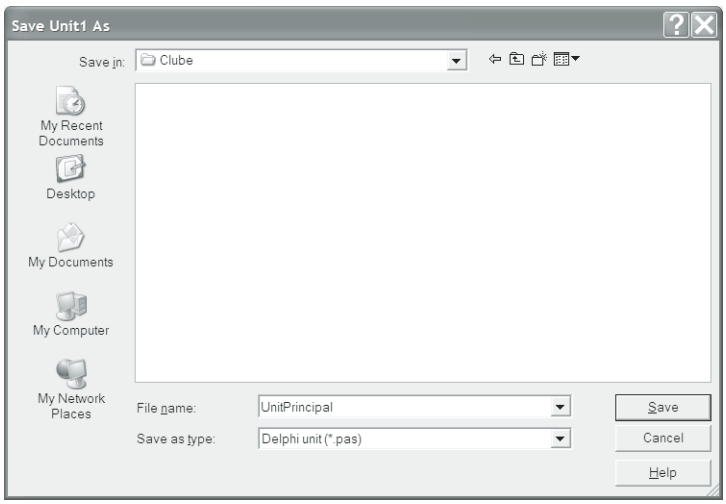


Figura 2.18: Salvando a primeira unidade de código.

Sempre que você mandar salvar as alterações de projetos, será solicitado que você atribua nomes a todas as units que foram criadas desde a última vez em que o projeto foi salvo, e que ainda não foram renomeadas (isto é, ainda estão com o nome default gerado pelo ambiente).

Após definir um nome para a primeira unidade de código e fechar a caixa de diálogo Save Unit1 As com o botão Salvar, será exibida a caixa de diálogo Save Project1 As (Figura 2.19), para que você defina o nome e a pasta do arquivo de projeto. Salve esse projeto com um nome fácil de se memorizar, mas que seja diferente dos nomes atribuídos aos arquivos de código (você não pode dar o mesmo nome a uma unit e a um arquivo de projeto). Nesse caso, adotamos o nome Clube, como mostra a figura a seguir.

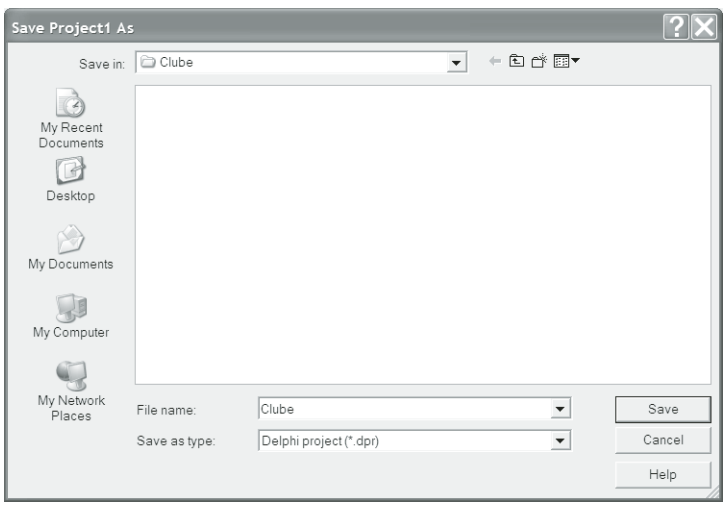


Figura 2.19: Salvando o arquivo de projeto.

Fechando um Projeto

Ao terminar de executar seu trabalho, você deve fechar o ambiente de desenvolvimento integrado. Antes disso, no entanto, você deve salvar as alterações realizadas no projeto.

Para salvar todas as alterações realizadas, basta executar um dos seguintes procedimentos:

1. Selecione o item Save All do menu File.

Ou:

1. Na caixa de ferramentas, selecione o ícone correspondente.

Para fechar o projeto, basta executar um dos seguintes procedimentos:

1. Selecione o item Close do menu File.

Ou:

1. Selecione o item Close All do menu File.

Se você alterar o projeto atual ou um dos arquivos de código e não salvá-lo antes de tentar executar os passos anteriores, será apresentada uma caixa de diálogo como as das Figuras 2.20 e 2.21, perguntando se as alterações devem ser salvas. Caso não tenha sido definido um nome para o projeto e para cada uma das suas unidades de código, também serão apresentadas as caixas de diálogo Save Unit1 As e Save Project1 As, descritas anteriormente.

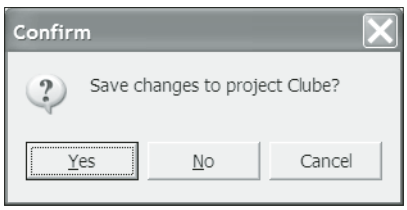


Figura 2.20: Confirmando se as alterações feitas no projeto devem ser salvas.

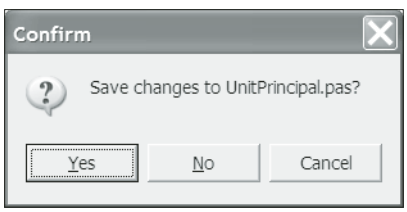


Figura 2.21: Confirmando se as alterações feitas no arquivo devem ser salvas.

MANIPULANDO GRUPOS DE PROJETOS

O Delphi 7 permite que você reúna vários projetos em uma nova entidade, denominada "Grupo de Projeto".

Essa característica é muito útil quando você possui vários projetos inter-relacionados e deseja manipulá-los simultaneamente no ambiente de desenvolvimento.

Para criar um novo grupo de projeto, você deve executar os seguintes procedimentos:

1. Selecione o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items (Figura 2.22).

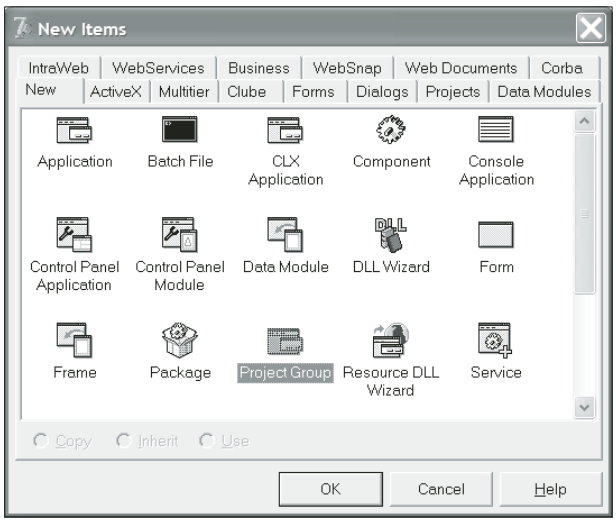


Figura 2.22: A caixa de diálogo New Items.

2. Selecione o item Project Group, na página New desta caixa de diálogo.
3. Clique em OK, para fechar a caixa de diálogo.

Será exibida a caixa de diálogo Project Manager, mostrada na figura a seguir.

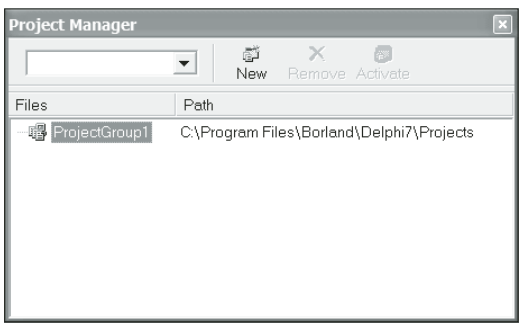


Figura 2.23: A caixa de diálogo Project Manager.

Nessa caixa de diálogo você pode:

- Adicionar um novo projeto ao grupo de projetos atual.
- Adicionar um projeto existente ao grupo de projetos atual.

Para adicionar um novo projeto ao grupo de projetos atual, você deve executar os seguintes procedimentos:

1. Selecionar o botão New na janela do Project Manager.

Ou:

1. Selecionar o botão direito do mouse com o nome do grupo de projetos selecionado, para exibir o menu pop-up dessa janela.
2. Selecionar o item Add New Project desse menu pop-up, como mostrado na figura a seguir. Será exibida a caixa de diálogo New Items, na qual você deve selecionar o item Application e o botão OK.

Repare que, após adicionar um novo projeto, este será exibido na caixa de diálogo Project Manager, como indicado na figura a seguir. Repare que o novo projeto (Project1) possui uma unit chamada Unit1, à qual correspondem um arquivo de código (chamado Unit1.pas) e um formulário (chamado Form1).

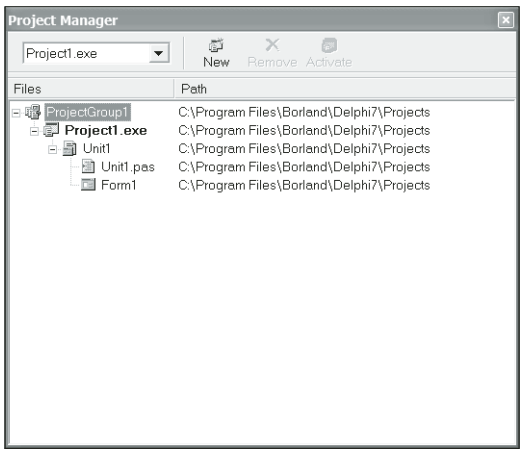


Figura 2.24: Adicionando o novo projeto (Project1) ao grupo de projetos atual.

Para adicionar um projeto existente ao grupo de projetos atual, você deve executar os seguintes procedimentos:

1. Selecionar o botão direito do mouse com o nome do grupo de projetos selecionado, para exibir o menu pop-up dessa janela.
2. Selecionar o item Add Existing Project desse menu pop-up, como mostrado na figura a seguir. Será exibida a caixa de Open Project, na qual você deverá selecionar o projeto a ser incluído no grupo de projetos atual e o botão Abrir.

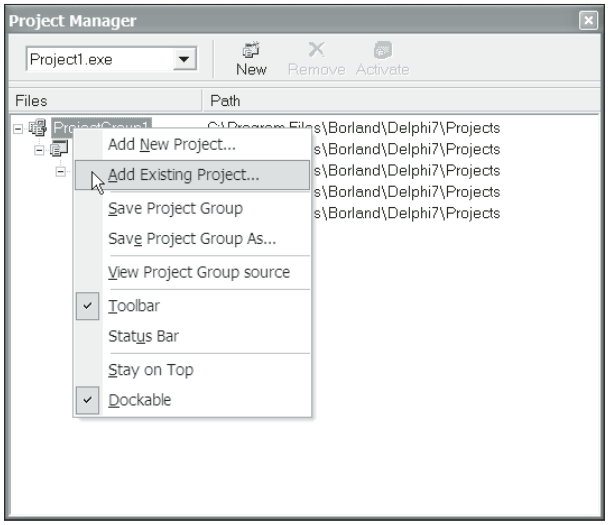


Figura 2.25: Adicionando um projeto existente ao grupo de projetos atual.

No Delphi 7, você pode adicionar ou remover unidades de código em um projeto simplesmente arrastando o seu ícone a partir do Windows Explorer para o local adequado na janela do Project Manager.

A figura a seguir apresenta a janela do gerenciador de projetos (Project Manager) após a inclusão de um novo projeto (Project1.exe) e um projeto existente (Clube.exe).

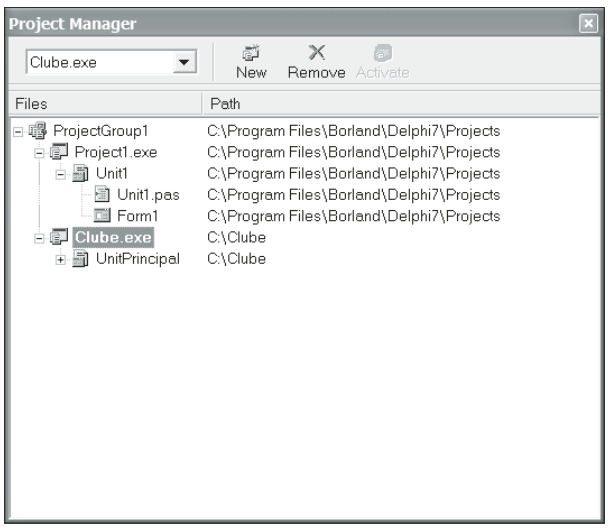


Figura 2.26: Manipulando vários projetos com o Project Manager.

Para remover um projeto do grupo de projetos atual, você deve executar os seguintes procedimentos:

1. Selecionar o projeto que se quer remover na janela do Project Manager.
2. Selecionar o botão Remove na janela do Project Manager.

Ou:

1. Selecionar o projeto que se quer remover na janela do Project Manager.
2. Selecionar o botão direito do mouse com o nome do grupo de projetos selecionado, para exibir o menu pop-up dessa janela.
3. Selecionar o item Remove Project desse menu pop-up. Será exibida a caixa de diálogo de confirmação mostrada na figura a seguir, na qual você deve selecionar o botão OK para finalizar esta tarefa.

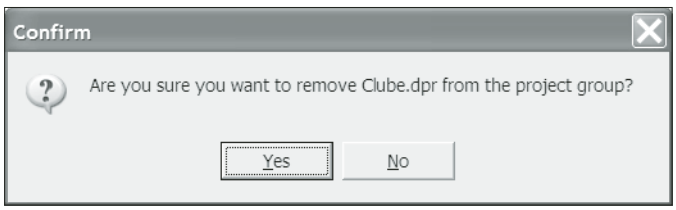


Figura 2.27: A caixa de diálogo de confirmação.

O Code Explorer

Conforme descrito anteriormente, inicialmente existe outra janela ancorada ao longo da borda esquerda da janela do Editor de Códigos. Essa janela (denominada Code Explorer) pode ser "destacada" do Code Editor, bastando, para isso, selecionar a barra horizontal situada no topo da janela e "arrastá-la" para fora da janela do Editor de Códigos.

A figura a seguir mostra a janela do Code Explorer, após ser destacada do Editor de Códigos.

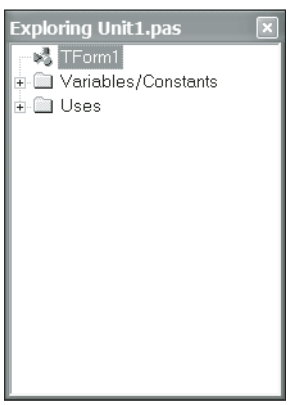


Figura 2.28: A janela do Code Explorer, destacada da janela do Editor de Códigos.

Você pode ancorar essa janela em qualquer janela do ambiente, como no Object Inspector, por exemplo.

Para isso, basta selecionar a barra horizontal superior da janela do Code Explorer com o botão esquerdo do mouse, arrastá-la e "soltá-la" na janela em que deseja ancorá-la.

A figura a seguir mostra a janela do Code Explorer "ancorada" no Object Inspector.

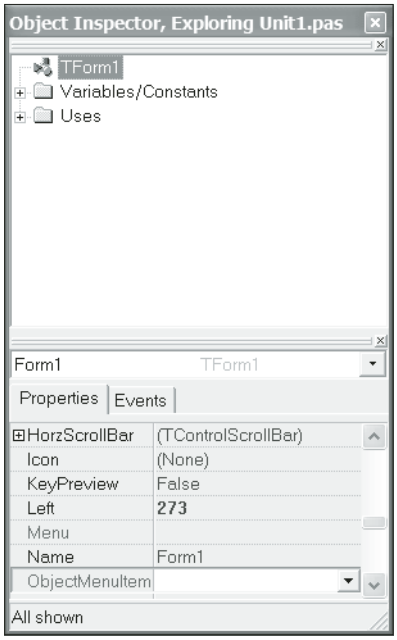


Figura 2.29: A janela do Code Explorer, ancorada na janela do Object Inspector.

Conforme descrito anteriormente, a janela do Code Explorer exibe uma árvore hierárquica que mostra os tipos, classes, propriedades, variáveis e rotinas globais, além dos nomes das units listadas na cláusula Uses da unit corrente.

O Code Explorer permite que se acesse o trecho de código no qual é definido um tipo de dado ou declarada uma função dando-se um duplo clique com o botão esquerdo do mouse sobre o item desejado na árvore hierárquica do Code Explorer.

Esse recurso é muito útil quando se está trabalhando com unidades de código extensas, nas quais se deseja localizar o trecho de código no qual é declarado um tipo, uma variável, ou é implementada uma função. Além disso, a janela do Code Explorer possui o recurso de pesquisa incremental, isto é, você pode localizar um item, digitando o seu nome quando a janela do Code Explorer possui o foco.

Desabilitando o Recurso de Ancoragem de Janelas

Existem situações em que você pode preferir mover uma janela pelo ambiente de desenvolvimento sem que esta possa ser "ancorada" em alguma outra.

Para desabilitar esse recurso, você deve executar os seguintes procedimentos:

1. Selecionar a janela para a qual deseja desabilitar esse recurso.
2. Selecionar o botão direito do mouse sobre essa janela, para exibir o seu menu pop-up.
3. Desmarcar o item Dockable desse menu pop-up.

Garantindo a Visibilidade de uma Janela

Existem situações em que você pode preferir manter uma janela permanentemente visível no ambiente de desenvolvimento, evitando a sua sobreposição por outras janelas.

Para habilitar esse recurso, você deve executar os seguintes procedimentos:

1. Selecionar a janela para a qual deseja habilitar esse recurso.
2. Selecionar o botão direito do mouse sobre essa janela, para exibir o seu menu pop-up.
3. Selecionar o item Stay on Top desse menu pop-up, como mostrado na figura a seguir.

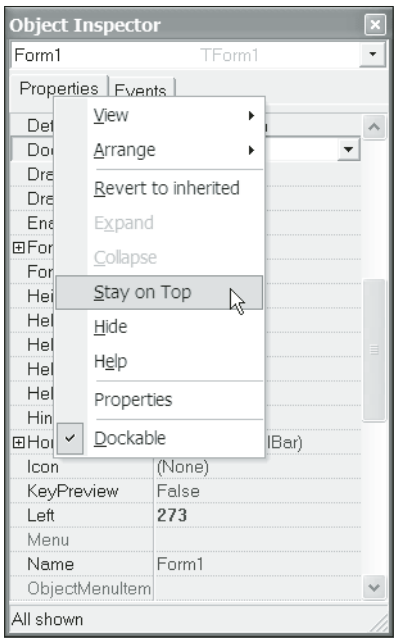
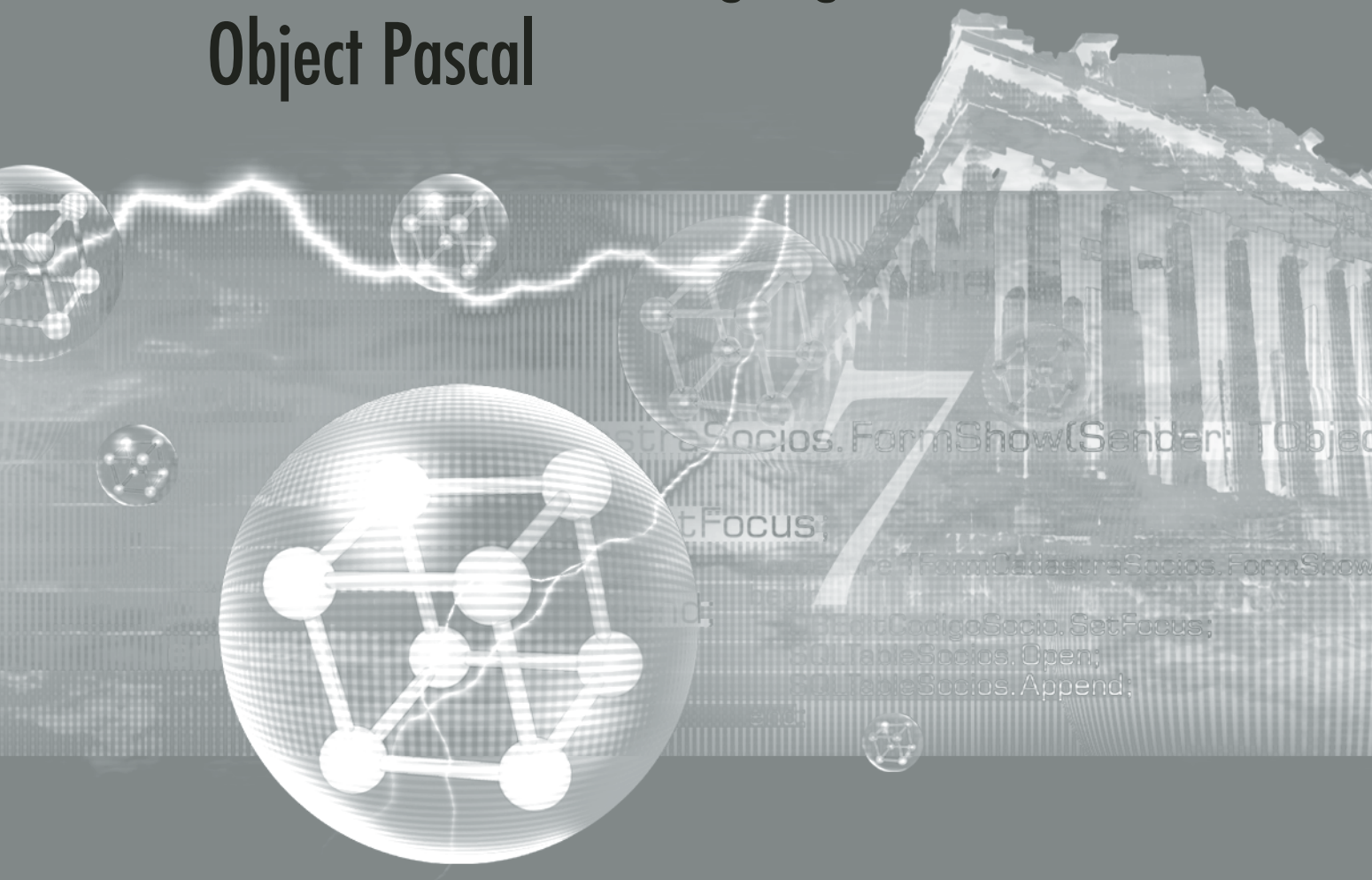


Figura 2.30: Selecionando o item Stay on Top do Object Inspector.

Capítulo 3

Fundamentos da Linguagem Object Pascal



Neste capítulo serão examinados o código gerado pelo ambiente de desenvolvimento do Delphi 7, e os conceitos de variáveis, classes e objetos da linguagem Object Pascal – a linguagem de programação empregada pelo ambiente de desenvolvimento do Delphi 7.

Os conceitos apresentados ajudarão a esclarecer algumas dúvidas que o leitor ainda possa ter após a leitura do capítulo anterior.

Fundamentos em: Estrutura de uma Unidade de Código (Unit)

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.

METODOLOGIA

- Apresentação e descrição dos elementos que compõem uma unit.

Examinando o Código de uma Unidade de Código (Unit) Gerado Pelo Delphi 7

No final do capítulo anterior, iniciamos o projeto do aplicativo-exemplo que será desenvolvido ao longo desta primeira parte do livro. Por enquanto, a aplicação – o projeto Clube – consiste em um único formulário e seu arquivo de código associado (que foi salvo com o nome UnitPrincipal.pas).

A seguir, apresentamos os arquivos de código gerados pelo Delphi 7, com os quais trabalharemos ao longo deste capítulo.

Arquivo de código gerado pela VCL:

```
unit UnitPrincipal;
interface
```

```
unit UnitPrincipal;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Arquivo de código gerado pela CLX:

```
unit UnitPrincipal;
interface

uses
  SysUtils, Types, Classes, Variants, QTypes, QGraphics, QControls, QForms,
  QDialogs, QStdCtrls;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.xfm}

end.
```

Inicialmente observa-se a palavra reservada *unit* seguida do nome do arquivo em que a unit está armazenada (já discutido no capítulo anterior).

Nos próximos tópicos, será detalhado o significado de cada uma das outras linhas de código, automaticamente geradas pelo Delphi 7, e serão apresentados os tipos de dados predefinidos da linguagem Object Pascal.

Examinando as Seções de uma Unit

Na linha seguinte à que define o nome da unit, tem-se a palavra-chave *interface*, que, juntamente com a palavra-chave *implementation*, define as duas principais seções de uma unit.

Na seção Interface são declarados os tipos de dados, classes, variáveis, funções e procedimentos que podem ser acessados por outras units.

Na seção Implementation são declarados os tipos de dados, classes, variáveis, funções e procedimentos que não podem ser acessados por outras units. Nessa seção também são implementadas as funções e procedimentos cujo cabeçalho é declarado na seção Interface.

Além dessas duas seções, você pode incluir, opcionalmente, mais duas, denominadas Initialization e Finalization. Na seção Initialization são incluídos comandos de atribuições de valores a variáveis, que devem ser processados assim que a aplicação for inicializada. A seção Finalization, por outro lado, armazena os comandos que devem ser executados quando a aplicação é finalizada (estas duas seções não são criadas automaticamente pelo Delphi 7).

Acessando Tipos e Variáveis Definidos em Outras Units

Após a palavra-chave *interface*, tem-se o seguinte trecho de código:

Numa unit gerada com o uso da VCL:

```
uses
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;
```

Numa unit gerada com o uso da CLX:

```
uses
  SysUtils, Types, Classes, Variants, QTypes, QGraphics, QControls, QForms,
QDialogs, QStdCtrls;
```

A palavra *uses* é outra palavra reservada da linguagem Object Pascal e define as units que serão utilizadas pela unit corrente. Repare que o ambiente já incluiu diversas units nessa cláusula uses. No caso da VCL, por exemplo, foi incluída a unit Windows (na qual estão armazenadas as declarações das funções da API do Windows). Se você remover a unit Windows da cláusula uses da unit, não terá mais acesso às funções da API do Windows.

Você também pode, opcionalmente, incluir uma cláusula uses na seção Implementation de uma unit. Isso evita problemas como referência circular e impede que as units referenciadas nessa cláusula uses sejam acessadas por outros usuários (lembre-se de que nada que é declarado na seção Implementation pode ser visto por outras unidades de código que usam essa unit).

Fundamentos em: Declaração de Variáveis

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.

METODOLOGIA

- Apresentação e descrição dos conceitos de variáveis, bem como dos tipos predefinidos da linguagem Object Pascal.

TÉCNICA

- Apresentação dos procedimentos necessários à declaração de variáveis e tipos na linguagem Object Pascal.

O Conceito de Variáveis

Em um programa de computador, dados são armazenados em variáveis. Na linguagem Object Pascal, a declaração de uma variável de um determinado tipo é feita na seção var de uma unit (ou de uma função ou procedure, no caso de se desejar criar uma variável local, conforme será visto posteriormente) e obedece à seguinte sintaxe:

```
var
    Nome_variável: Tipo_da_variável;
```

Você pode declarar simultaneamente mais de uma variável de um mesmo tipo. Nesse caso, os nomes das variáveis devem vir separados por vírgulas, obedecendo à seguinte sintaxe:

```
var
   Nome_variável1, Nome_variável2,..., Nome_variáveln: Tipo_da_variável;
```

Nos trechos de código exemplificados anteriormente, a palavra reservada *var* (que já existe na unit gerada) foi incluída apenas para tornar mais claro o contexto em que uma variável é declarada.

Os nomes das variáveis devem começar com uma letra ou um caractere sublinhado (_), seguidos por uma seqüência de letras, dígitos ou caracteres sublinhados, e não podem conter espaços em branco ou caracteres especiais como $,%, acentuação e operadores aritméticos.

Atribuindo um Valor a uma Variável

Para atribuir um valor a uma determinada variável, você deve usar o comando de atribuição da linguagem Object Pascal, que apresenta a seguinte sintaxe:

```
nome_variavel:= valor;
```

Nesse caso, conforme descrito anteriormente, o sinal de igualdade precedido de dois-pontos é denominado "operador de atribuição".

O programador novato não deve estranhar a utilização da seguinte linha de código:

```
X := X + 1;
```

Evidentemente, se nesse caso o sinal de igual estivesse realmente representando a igualdade entre dois valores, essa equação seria matematicamente absurda.

Entretanto, se considerarmos que, nesse caso, esse sinal está funcionando como um operador de atribuição, a expressão anterior passa a ter sentido, pois, nesse caso, estamos atribuindo à variável X o valor armazenado nessa variável, acrescido de uma unidade. Essa expressão deve ser entendida da seguinte maneira: "Obtenha o valor armazenado na variável X, acrescente uma unidade a esse valor e atribua o resultado à variável X".

Conseqüentemente, analisando-a como um comando de atribuição, essa expressão passa a ter sentido.

Tipos de Dados Predefinidos na Linguagem Object Pascal

Na linguagem Object Pascal estão definidos os seguintes tipos de dados predefinidos (tipos de variáveis):

Tipos de Variáveis Inteiras

Tipo	Faixa de Valores	Formato
Integer	–2147483648..2147483647	32 bits
Cardinal	0..4294967295	32 bits, sem sinal
Shortint	–128..127	8 bits

Tipo	Faixa de Valores	Formato
Smallint	–32768..32767	16
Longint	–2147483648..2147483647	32
Int64	–2^63..2^63–1	64
Byte	0..255	8 bits, sem sinal
Word	0..65535	16 bits, sem sinal
Longword	0..4294967295	32 bits, sem sinal

Tipos de Variáveis Reais

Tipo	Faixa de Valores	Dígitos Significativos	Tamanho (Bytes)
Real48	2.9 x 10^–39 .. 1.7 x 10^38	11–12	6
Single	1.5 x 10^–45 .. 3.4 x 10^38	7–8	4
Double	5.0 x 10^–324 .. 1.7 x 10^308	15–16	8
Extended	3.6 x 10^–4951 .. 1.1 x 10^4932	19–20	10
Comp	–2^63+1 .. 2^63 –1	19–20	8
Currency	–922337203685477.5808.. 922337203685477.5807	19–20	8

O tipo genérico Real é equivalente a double.

Tipos de Variáveis Booleanas

Tipo	Faixa de Valores
Boolean	False ou True (0 ou 1)
ByteBool	*
WordBool	*
LongBool	*

(*) Esses tipos são definidos apenas por questão de compatibilidade com outras linguagens de programação. Uma variável booleana pode assumir apenas os valores ordinais 0 e 1, ao passo que os demais tipos podem assumir quaisquer valores ordinais. Nesses casos, a expressão será falsa se seu valor ordinal for 0, e verdadeira quando esse valor for diferente de 0 (como ocorre nas linguagens C e C++).

Tipos de Variáveis Para Manipulação de Caracteres

Na linguagem Object Pascal, caracteres e strings literais são delimitados por aspas simples. Para a manipulação de textos, compostos de um ou mais caracteres, a linguagem Object Pascal apresenta os tipos de variáveis listados a seguir:

- Char: Tipo alfanumérico que pode armazenar um caractere ASCII.
- AnsiChar: Tipo alfanumérico que pode armazenar um caractere ASCII (8 bits).
- WideChar: Tipo alfanumérico que pode armazenar um caractere Unicode (16 bits).
- ShortString: Tipo alfanumérico que tem como conteúdo uma cadeia de até 255 caracteres.
- AnsiString: Tipo alfanumérico que tem como conteúdo uma cadeia ilimitada de caracteres (Char).
- WideString: Tipo alfanumérico que tem como conteúdo uma cadeia ilimitada (na verdade há um limite, definido pela memória disponível) de caracteres (WideChar).

A linguagem Object Pascal suporta ainda as strings de terminação nula, cujo término é indicado pelo caractere #0 (NULL). Esse tipo de string deve ser utilizado apenas em casos em que for realmente necessário, em chamadas das funções que necessitam de parâmetros do tipo PChar (ponteiro para caractere) como, por exemplo, quando se utilizam, no caso de se estar usando a VCL, chamadas a funções da API do Windows ou funções exportadas por DLLs criadas em outras linguagens para este ambiente.

Você pode manipular esse tipo de dado criando arrays do tipo Char, AnsiChar e WideChar, nas quais o caractere NULL será armazenado na posição que indica o final da string. Outra alternativa consiste em utilizar as funções de manipulação de string, a serem vistas posteriormente.

Tipos de Variáveis Para Manipulação de Arquivos

Para a manipulação de arquivos, a linguagem Object Pascal apresenta a palavra reservada *File*, que pode ser usada para criar um tipo a ser utilizado para representar um arquivo.

O tipo File consiste em uma seqüência linear de valores de um determinado tipo (que pode ser qualquer tipo de variável, exceto o próprio tipo File), incluindo-se uma classe ou um tipo estruturado definido pelo programador.

Para criar uma variável para representar um arquivo destinado a armazenar valores de um determinado tipo, deve-se incluir uma linha de código com a seguinte sintaxe, na seção var de uma unit (ou de procedure ou função, no caso de variável local):

```
nome_da_variável: File of Tipo;
```

Por exemplo, para criar uma variável denominada arq_real, para representar um arquivo destinado a armazenar números reais, deve ser usada a seguinte declaração:

```
arq_real: File of Real;
```

Para armazenar texto no formato ASCII, existem os tipos predefinidos Text e TextFile. Os dois são equivalentes.

Nesse caso, a declaração correspondente seria:

```
arq_text: TextFile;
```

Existem funções específicas para manipulação de arquivos, que serão abordadas ainda neste capítulo, após os tópicos correspondentes às funções e procedimentos.

Tipo Genérico de Variáveis

A linguagem Object Pascal apresenta um tipo genérico de variável, denominado Variant, e que será descrito a seguir.

O tipo Variant pode armazenar qualquer tipo predefinido de variável durante a execução do aplicativo. Esse tipo foi introduzido na versão 2.0 do Borland Delphi, mas apresenta como desvantagens um maior consumo de memória para armazenar um valor e de tempo para executar uma operação. Deve ser utilizado quando não se conhece, *a priori*, o tipo de valor que será armazenado na variável.

Cabe a você, desenvolvedor, considerar as vantagens e desvantagens de se utilizar variáveis desse tipo.

Comentários

Os comentários constituem porções de texto cuja finalidade é documentar o código de um programa, sendo eventualmente empregado como recurso para a depuração de programas. A linguagem Object Pascal admite dois tipos de comentários:

Comentários de uma única linha

Este tipo de comentário é identificado por duas barras inclinadas. A partir destas duas barras, tudo o que for escrito na mesma linha será tratado como um comentário.

Exemplo:

```
// isto é um comentário de uma única linha
```

Comentários de múltiplas linhas

A linguagem Object Pascal admite dois tipos de comentários de múltiplas linhas. A primeira opção consiste em colocar o texto entre chaves de abertura – { – e de fechamento – }.

Exemplo:

```
{ isto é um comentário
de múltiplas linhas }
```

A segunda opção consiste em colocar o texto entre os seguintes identificadores: Parênteses-asterisco para início do comentário e asterisco-parênteses para o seu término.

```
(* isto é um comentário
de múltiplas linhas *)
```

Definindo Novos Tipos de Dados

Logo após a cláusula uses da seção Interface de uma unit, aparece a palavra-chave *type*.

Esse identificador é usado para definir novos tipos de dados, conforme será descrito nos tópicos a seguir.

Tipos de Dados Enumerados

Um tipo de dado enumerado consiste em um grupo com um número finito de elementos, separados por vírgulas e inseridos entre um par de parênteses.

Por exemplo, para se criar um tipo de dado enumerado chamado semana, cujos elementos são os dias da semana, deve-se incluir a seguinte linha de código após a palavra-chave *type*:

```
semana = (Domingo, Segunda, Terca, Quarta, Quinta, Sexta, Sabado);
```

Esse tipo de dado ainda apresenta uma característica interessante: seus elementos têm uma relação de ordem, isto é, Domingo é menor que Segunda, que é menor que Terça, e assim por diante.

Podemos, então, declarar uma variável denominada dia, do tipo semana, incluindo a sua declaração após a palavra-chave *var*:

```
var
   dia: semana;
```

Essa variável pode, portanto, receber qualquer um dos valores definidos no seu tipo, como, por exemplo:

```
dia:= Quarta;
```

Conjuntos

Um conjunto é muito parecido com um tipo de dado enumerado, mas, nesse caso, a ordem dos elementos não é importante. Além disso, uma variável de um tipo definido como um conjunto pode armazenar vários elementos de um conjunto (podendo inclusive não armazenar qualquer dos elementos – caso em que se obtém um conjunto vazio).

Para criar um tipo de dado como um conjunto, você deve incluir, após a palavra reservada *type*, uma linha de comando com a seguinte sintaxe:

```
nome_do_tipo = set of tipo_já_definido;
```

A palavra reservada *set* indica que está sendo criado um conjunto baseado em um tipo ordinal já definido (pode ser um dos tipos fundamentais da linguagem Object Pascal, ou um tipo definido pelo desenvolvedor).

Por exemplo, para criar um tipo conjunto denominado letras, poderíamos considerar o seguinte trecho de código:

```
type
     letras = set of Char;
```

Podemos, então, declarar uma variável denominada consoante, do tipo letras, incluindo a sua declaração após a palavra-chave *var,* como exemplificado a seguir.

```
var
   consoante: letras;
```

A atribuição de valores a uma variável definida como um conjunto é feita colocando-se os elementos do conjunto entre colchetes, como mostrado a seguir.

```
consoante:= ['b','c', 'd', 'f', 'g', 'h', 'j', 'l', 'm', 'n', 'p', 'q', 'r', 's', 't', 'v',
'x', 'z'] ;
```

Para criar um conjunto vazio, basta incluir o seguinte trecho de código:

```
letras:= [ ];
```

Você pode realizar operações sobre conjuntos, tal qual se faz na matemática.

Apresentamos, a seguir, as operações definidas para o tipo Set:

Operador	Significado	Exemplo
+	união	Letras1 + Letras2 ou ['a', 'b'] + ['c', 'd']
–	diferença	Letras1 – Letras2 ou ['a', 'b'] – ['c', 'd']
*	interseção	Letras1 * Letras2 ou ['a', 'b'] * ['a', 'e']
<=	subconjunto	Letras1 <= Letras2 ou ['a', 'b'] <= ['a', 'd']
>=	superconjunto	Letras1 >= Letras2 ou ['a', 'b'] >= ['a', 'd']
=	igualdade	Letras1 = Letras2 ou ['a', 'b'] = ['a', 'd']
<>	desigualdade	Letras1 <> Letras2 ou ['a', 'b'] <> ['a', 'd']
in	pertinência	char in Letras1 ou 'a' in ['a', 'd']

A operação de união retorna um conjunto cujos elementos estão contidos em qualquer dos operandos.

A operação de diferença retorna um conjunto cujos elementos estão contidos no primeiro operando, mas não estão presentes no segundo.

A operação de interseção retorna um conjunto cujos elementos estão obrigatoriamente contidos simultaneamente nos dois operandos.

A operação de subconjunto corresponde ao "está contido" que aprendemos na matemática colegial, e retorna true ou false, conforme o primeiro operando seja ou não um subconjunto do segundo operando, isto é, se todo e qualquer elemento do primeiro operando está presente no segundo.

A operação de superconjunto corresponde ao "contém" que aprendemos na matemática colegial, e retorna true ou false, conforme o primeiro operando seja ou não um superconjunto do segundo operando, isto é, se todo e qualquer elemento do segundo operando está presente no primeiro.

A operação de igualdade retorna true ou false. Retorna true se todo elemento do primeiro operando estiver presente no segundo e vice-versa, retornando false em caso contrário.

A operação de desigualdade retorna true ou false. Retorna true se ao menos um elemento do primeiro operando não estiver presente no segundo e vice-versa, retornando false em caso contrário.

O operador in verifica se o elemento representado pelo primeiro operando está presente no conjunto representado pelo segundo operando, retornando false em caso contrário.

Variáveis Compostas

A linguagem Object Pascal permite que se definam tipos compostos de variáveis, denominados registros, aos quais pertence um conjunto de variáveis de tipos distintos. Isso é feito mediante a inclusão de um trecho de código na seção type de uma unit, e que apresenta a seguinte sintaxe:

```
nome_do_tipo = Record
    variável1: primeiro_tipo;
    variável2: segundo_tipo;
......
    variáveln: n-ézimo_tipo;
end;
```

As variáveis variável1, variável2, ..., variáveln contidas no registro são denominadas campos do registro. Podem ser de qualquer tipo, inclusive objetos de classes definidas no ambiente ou pelo próprio desenvolvedor (a definição de uma classe será apresentada nos próximos tópicos).

Para declarar uma variável desse tipo, devemos proceder da mesma forma que faríamos se quiséssemos declarar uma variável de qualquer tipo predefinido, bastando que se inclua, após a palavra reservada *var*, uma linha de código como:

```
Nome_variável: nome_do_tipo;
```

Para acessar um campo de uma variável composta, devemos usar a chamada notação de ponto, que pode ser descrita como:

```
nome_da_variavel.nome_do_campo;
```

Por exemplo, para criar um tipo de variável composta chamada materia, poderíamos definir o seguinte trecho de código, após a palavra reservada *type*:

```
materia= Record
     Livro_Texto: string;
     carga_horaria: integer;
end;
```

Podemos, então, declarar uma variável chamada matematica, do tipo materia, incluindo a sua declaração após a palavra reservada *var*:

```
var
    matematica: materia;
```

Para atribuir valores aos seus campos, basta incluir as seguintes linhas de código:

```
matematica.Livro_Texto:= 'Matematica para Principiantes';
matematica.carga_horaria:= 75;
```

Vetores (Arrays)

Os vetores ou arrays são grupos de variáveis com mesmo nome, mas diferenciados por um índice, sendo úteis para armazenar uma grande quantidade de dados de um mesmo tipo.

Para criar um tipo de dado como um vetor, você deve incluir, após a palavra reservada *type*, uma linha de comando com a seguinte sintaxe:

```
nome_do_tipo: array[i1..i2] of tipo_da_variável;
```

Onde i1 e i2 determinam os valores mínimo e máximo do índice, respectivamente.

Por exemplo, para criar um tipo chamado letras como um vetor, poderíamos definir o seguinte trecho de código:

```
type
    letras = array[1..23] of Char;
```

Podemos, então, declarar uma variável denominada alfabeto, do tipo letras, incluindo a sua declaração após a palavra reservada *var*, como mostra o trecho de código a seguir.

```
var
    alfabeto: letras;
```

A atribuição de valores a uma variável definida como um vetor é feita independentemente para cada um dos seus elementos, indicando-se o seu índice entre colchetes, como mostramos a seguir.

```
alfabeto[1]:= 'a';
alfabeto[2]:= 'b';
alfabeto[3]:= 'c';
```

A linguagem Object Pascal também suporta diretamente o conceito de arrays dinâmicos (cujas dimensões podem ser alteradas durante a execução do aplicativo). A declaração de um array dinâmico é feita mediante a inclusão de uma linha de código que apresenta a seguinte sintaxe:

```
var
   A : array of integer;    // Declara um array unidimensional de inteiros.
   B : array of array of real; // Declara um array bidimensional de números reais.
   C : array of array of array of char; // Declara um array tridimensional de caracteres.
```

A simples declaração não aloca memória para o array, o que é feito mediante uma chamada ao procedimento SetLength.

Esse procedimento, declarado na unit System, recebe como parâmetros o nome do array e as suas novas dimensões.

Poderíamos, por exemplo, alocar memória para os arrays descritos anteriormente mediante a inclusão do seguinte trecho de código:

```
SetLength(A,20);
SetLength(B,20,5);
SetLength(C,20,10,15);
```

É importante deixar claro para o leitor que, para arrays dinâmicos, o índice de cada dimensão começa em 0, e não em 1, como poderia indicar a nossa intuição.

Para atribuir um valor a um elemento do array multidimensional C, deve-se utilizar a seguinte linha de código:

```
C[4,3,7]:= 'k';
```

O procedimento SetLength também pode ser usado para definir o número de caracteres de uma string definida como ShortString e para redimensionar um array durante a execução do aplicativo (nesse caso, os valores existentes no array são preservados quando sua dimensão aumenta).

Para liberar a memória alocada para um array, existem duas alternativas:

- Atribuir o valor nil à variável que representa o array.
- Passar o nome da variável que define o array em uma chamada ao procedimento Finalize (também definido na unit System).

Operadores Aritméticos

Na linguagem Object Pascal, você pode realizar operações aritméticas sobre variáveis, e os seguintes operadores estão disponíveis:

Operador	Significado
*	Multiplicação
/	Divisão (entre duas variáveis reais)
+	Soma (ou concatenação, no caso de strings)
-	Subtração
div	Divisão (entre duas variáveis inteiras)
mod	Resto da divisão entre duas variáveis inteiras

Você pode realizar uma operação aritmética entre duas variáveis e atribuir o resultado a uma terceira variável, mediante a inclusão de uma linha de código com a seguinte sintaxe:

```
variavel3:= variável1 op variavel2;
```

Onde op é um dos operadores aritméticos descritos anteriormente.

É importante respeitar a precedência dos operadores, que pode ser alterada mediante a utilização de parênteses. A seguir apresentamos a ordem de precedência dos diversos operadores:

Operador	Ordem de Precedência
@, not	1
*, /, div, mod, and, shl, shr, as	2
+, –, or, xor	3
=, <>, <, >, <=, >=, in, is	4

Os operadores + e -, quando aplicados a um único operando (como operadores unários, indicando o sinal de um número ou variável), possuem ordem de precedência igual a 1.

Alguns desses operadores ainda não foram apresentados, e serão vistos ao longo do livro. O operador @, por exemplo, é utilizado para obter o endereço de uma variável (e que provavelmente será atribuído a uma variável do tipo ponteiro).

Operadores com maior precedência (os primeiros da tabela) são executados antes dos de menor precedência. Operadores com mesma ordem de precedência em uma expressão são executados da esquerda para a direita.

A expressão seguinte, por exemplo, resultaria no valor 28:

```
X := 8 + 5 * 4;
```

Caso você queira que a adição seja executada antes da multiplicação, deve incluir parênteses para alterar a ordem de precedência, como a seguir (o resultado, nesse caso, seria a atribuição do valor 52 à variável X):

```
X := (8 + 5) * 4;
```

Tipos Ordinais

Definem-se como ordinais os tipos de variáveis cujos valores têm uma relação de ordem entre si.

Dentre os tipos ordinais podem-se destacar os tipos integer e char, por exemplo.

Entre duas variáveis de um mesmo tipo ordinal podem-se estabelecer comparações do tipo menor que (<), maior que (>), etc.

Existem funções específicas para os tipos ordinais, que serão abordadas ainda neste capítulo, após os tópicos correspondentes às funções e procedimentos.

Escopo e Tempo de Vida das Variáveis

Define-se como escopo de uma variável o conjunto de trechos de código nos quais a variável é visível (isto é, pode ser acessada).

Variáveis Locais

Na linguagem Object Pascal, quando uma variável for declarada dentro de uma função ou procedimento, ela somente será visível dentro da própria rotina (procedimento ou função), isto é, terá escopo local. Se, em outra função ou procedimento, você declarar uma variável com o mesmo nome, uma não tomará conhecimento da existência da outra, não havendo relação alguma entre elas, ou qualquer tipo de interferência. Conseqüentemente, os dados tratados dentro de um procedimento ou função estão protegidos contra intervenções externas.

Quando uma variável é definida em uma função ou procedimento, diz-se que essa variável tem o escopo da função ou procedimento (a variável é local à função ou procedimento), excetuando-se o caso em que a variável é definida como um parâmetro que é passado por referência (precedido pela palavra reservada *var*).

A criação de variáveis locais a procedimentos será descrita no tópico correspondente.

Variáveis Globais a uma Unidade de Código

Existem situações em que é necessário compartilhar dados entre funções ou procedimentos dentro de uma mesma unidade de código (unit). Você precisa, nesse caso, de variáveis que sejam acessíveis (visíveis) em toda a unidade de código (unit). Essas variáveis são ditas globais à unit e devem ser declaradas após a palavra reservada *var* na seção Interface da unit (antes da palavra reservada implementation).

Quando você quiser que uma variável declarada com o escopo de uma unit seja acessada por outra unit, essa última unit deve incluir o nome da primeira (que contém a declaração da variável) em sua cláusula uses.

Dê preferência às variáveis locais em seus programas, que, dessa maneira, ficarão mais estruturados, legíveis e com dados protegidos contra alterações acidentais.

Precisando compartilhar dados entre sub-rotinas ou procedimentos, dê preferência às variáveis com escopo de uma unit. Só permita que uma unit acesse os dados das variáveis declaradas em outra quando isso for estritamente necessário e lembre-se de que, se existirem duas variáveis com o mesmo nome e escopos diferentes, todas as referências ao nome da variável serão referências à variável declarada localmente.

Criação de Variáveis Globais a uma Aplicação

Variáveis globais são aquelas que são acessíveis (visíveis) em todas as units, funções e procedimentos. Para declarar um conjunto de variáveis globais a uma aplicação, proceda da seguinte forma:

1. Crie uma nova unit para a sua aplicação.
2. Inclua o nome dessa unit na cláusula uses de todas as outras units.

Pronto! As variáveis declaradas na seção Interface dessa unit serão acessadas em todo o código da aplicação. É recomendável que você salve essa unit com um nome fácil de ser memorizado, como global.pas, por exemplo.

O tópico seguinte mostra como adicionar uma unit para armazenar as variáveis globais de uma aplicação.

Adicionando uma Nova Unit ao Projeto Para Armazenamento de Variáveis Globais

Um projeto de aplicação não é composto apenas por formulários, mas também inclui unidades de código (as units). Quando você cria um novo formulário, o ambiente cria automaticamente uma unit associada a esse formulário. Nada impede, no entanto, que você inclua no projeto da sua aplicação novas unidades de código que sejam independentes de formulários. Essas unidades de código podem ser usadas para armazenar tipos especiais de classes, funções, procedimentos e variáveis.

No caso da VCL, por exemplo, as units Windows.pas e System.pas definem uma quantidade enorme de funções e não estão associadas a nenhum formulário.

Para criar uma unit independente, proceda da seguinte forma:

1. Selecione a opção New/Other do menu File do Delphi 7. Será exibida a caixa de diálogo New Items (Figura 3.1).
2. Selecione a guia New.
3. Selecione o ícone correspondente à opção Unit.
4. Clique no botão OK, para fechar a caixa de diálogo. Será criada uma nova unit (independente de qualquer formulário), na qual podem ser definidas as novas classes, funções e procedimentos.

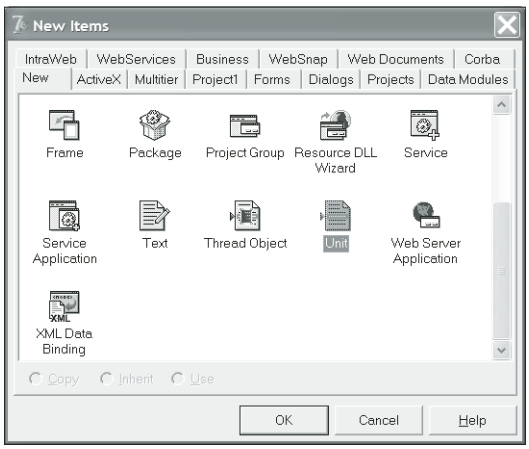


Figura 3.1: A caixa de diálogo New Items do Delphi 7.

Fundamentos em: Blocos de Comandos, Estruturas Condicionais e de Repetição

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.

METODOLOGIA

- Apresentação e descrição dos elementos de sintaxe necessários à criação dos blocos de comandos e das estruturas condicionais e de repetição.

TÉCNICA

- Apresentação e descrição dos procedimentos necessários à criação dos blocos de comandos e das estruturas condicionais e de repetição da linguagem Object Pascal.

Alterando o Fluxo de Execução do Programa

Desde os primórdios da computação, as estruturas condicionais e de repetição têm sido fundamentais para que o fluxo dos sistemas possa variar em função de determinadas condições.

Essas estruturas estão presentes na totalidade das linguagens de programação, e a linguagem Object Pascal não seria uma exceção.

Blocos de Comandos

Na linguagem Object Pascal, um bloco de comandos é constituído por um conjunto de linhas de código, que começa com a palavra reservada *begin* e termina com a palavra reservada *end*, seguida de um ponto-e-vírgula (;).

Um bloco de comandos é definido pela sintaxe:

```
begin
        {Instruções do Bloco de comandos}
end;
```

A palavra reservada *end* só não deve ser seguida por um ponto-e-vírgula nas seguintes situações:

- Antes da palavra reservada *else*, em uma estrutura condicional do tipo if-then-else (a ser abordada nos próximos tópicos).
- No final de uma unit (o end que encerra uma unit é seguido por um ponto).

Estruturas Condicionais

Existem situações em que se deseja executar um trecho de código, apenas se uma determinada condição for verdadeira, e um outro trecho de código, caso a condição testada seja falsa.

Na linguagem Object Pascal, isso é obtido utilizando-se as estruturas condicionais if-then-else e case of, descritas a seguir.

Estrutura Condicional if-then-else

A estrutura condicional if-then-else da linguagem Object Pascal apresenta a seguinte sintaxe:

```
if (condição)
then
   begin
        {Bloco de comandos executados se a condição for verdadeira}
   end
else
   begin
        {Bloco de comandos executados se a condição for falsa}
   end;
```

Caso você não queira executar qualquer comando se a condição for falsa, basta suprimir o trecho de código correspondente à cláusula else, como mostrado no trecho de código a seguir.

```
if (condição)
then
   begin
        {Bloco de comandos executados se a condição for verdadeira}
   end;
```

Nunca coloque um ponto-e-vírgula antes da palavra reservada *else*. Isso gerará um erro de compilação.

Nos casos em que um bloco de comando é formado por uma única linha de código, podem-se suprimir as palavras *begin* e *end*, como mostram os trechos de código a seguir.

```
if (condição)
then
{Comando executado se a condição for verdadeira}
else
{Comando executado se a condição for falsa};
```

e:

```
if (condição)
then
        {Comandos executados se a condição for verdadeira}
```

Estrutura Condicional Case Of

A estrutura condicional case of da linguagem Object Pascal tem a seguinte sintaxe:

```
case <expressão> of
      Valor_1:
           <Bloco de comandos>
      Valor_2:
           <Bloco de comandos>
      ........................
      Valor_n:
           <Bloco de comandos>
      else:
           <Bloco de comandos>
end;
```

Nesse caso, se a expressão testada for igual a um dos valores especificados (Valor_1, Valor_2,..., Valor_n), será executado o bloco de comandos a ele correspondente. Caso nenhum desses valores seja igual ao definido pela expressão testada, o bloco de comandos correspondente à palavra reservada *else* será executado.

A expressão avaliada em uma estrutura condicional case of deve ser de um tipo ordinal.

Testes Condicionais

Nos tópicos anteriores foram apresentadas as estruturas condicionais da linguagem Object Pascal. Nessas estruturas, a condição a ser testada deve retornar o valor False (falso) ou True (verdadeiro). Quando a condição a ser testada é uma variável booleana, a sua verificação é imediata.

Existem situações, no entanto, em que o resultado da condição advém de uma relação entre dois operandos. Quando for esse o caso, devem-se usar os operadores relacionais da linguagem Object Pascal, mostrados nos tópicos a seguir.

Os Operadores Relacionais

A linguagem Object Pascal tem os seguintes operadores relacionais:

Operador	Finalidade
= (igual):	Usado para testar se dois valores são iguais.
<> (diferente):	Usado para testar se dois valores são diferentes.
< (menor):	Usado para testar se um valor é menor do que outro.
<= (menor ou igual):	Usado para testar se um valor é menor ou igual a outro.
> (maior):	Usado para testar se um valor é maior do que outro.
>= (maior ou igual):	Usado para testar se um valor é maior ou igual a outro.
in (pertinência):	Verifica se um valor pertence ou não a um conjunto.

O trecho de código a seguir, por exemplo, compara o valor de duas variáveis e exibe uma mensagem em função do resultado obtido.

```
if (a < b)
then
     Showmessage(´a é menor que b´)
else
     Showmessage(´a é maior ou igual a b´);
```

Estruturas de Repetição

Quando um mesmo tipo de comando (ou bloco de comandos) precisa ser executado repetidamente, pode-se economizar a sua codificação usando-se uma das estruturas de repetição da linguagem Object Pascal.

A linguagem Object Pascal tem os seguintes tipos de estruturas de repetição:

- Laços For
- Laços While
- Laços Repeat

Laços For

Essa estrutura de repetição é bastante útil quando se deseja que a execução de um bloco de comandos seja repetida um número predeterminado de vezes. Essa estrutura apresenta a seguinte sintaxe:

```
for contador:= valor_inicial to valor_final do
<bloco de comandos>
```

Onde:

- contador: É uma variável ordinal enumerável, normalmente inteira.
- valor_inicial: É o valor inicial do contador, geralmente um número inteiro.
- valor_final: É o valor final assumido pelo contador, geralmente um número inteiro.
- bloco de comandos: É uma seqüência de comandos que começa com a palavra reservada *begin* e termina com a palavra reservada *end*.

Desse modo, a variável inteira contador varia desde o valor inicial até o valor final, em incrementos unitários e, para cada incremento, executa o bloco de comandos que se inicia na linha seguinte.

Caso se queira que o contador assuma valores decrescentes, deve-se usar a seguinte sintaxe:

```
for contador:= valor_inicial downto valor_final do
<bloco de comandos>
```

Nesse caso, evidentemente, o valor inicial deve ser superior ao valor final.

O contador pode ser qualquer tipo ordinal enumerável. O código abaixo, por exemplo, é perfeitamente válido (embora possa não ser muito útil).

```
for contador:= 'a' to 'z' do
     Showmessage(contador);
```

A variável contador, nesse caso, deve ser declarada como sendo do tipo char.

Laços While

Essa estrutura de repetição é bastante útil quando se deseja que a execução de um bloco de comandos seja repetida enquanto uma determinada condição for verdadeira.

Essa estrutura de repetição apresenta a seguinte sintaxe:

```
while <condição> do
<bloco de comandos>
```

Observe que, se a condição for falsa na primeira vez em que a estrutura de repetição for acessada, o bloco de comandos não será executado nenhuma vez.

Essa estrutura de repetição é muito útil quando se quer, por exemplo, ler as informações a partir de um arquivo, pois, nesse caso, não se sabe previamente quantas iterações serão necessárias para ler todo o seu conteúdo. O seguinte trecho de código pode ser usado:

```
while not Eof(F1) do
begin
        Read(F1, Ch);
        Write(F2, Ch);
end;
```

Onde F1 e F2 são variáveis de arquivo e Eof é uma função que retorna o valor True, se for alcançado o final do arquivo, e False, em caso contrário. Portanto, enquanto houver registros a serem lidos no arquivo, isto é, enquanto a condição Eof(F1) for falsa (e not Eof(F1) for verdadeira), o bloco de código será executado, lendo dados de F1 e gravando-os em F2.

Laços Repeat

Essa estrutura de repetição é bastante útil quando se pretende que a execução de um bloco de comandos seja repetida enquanto uma determinada condição for verdadeira, mas se impõe que esse bloco de comandos seja executado ao menos uma vez. Nesse caso, ao contrário do que ocorre nos laços While,

a condição é testada após a primeira execução do bloco de comandos, garantindo que ele seja executado ao menos uma vez.

Essa estrutura de repetição apresenta a seguinte sintaxe:

```
repeat
<bloco de comandos>
until condição;
```

Condições Compostas

Existem situações em que uma condição a ser testada é, na realidade, uma combinação de duas ou mais condições.

Normalmente, uma condição composta é testada usando-se os operadores lógicos da linguagem Object Pascal, apresentados no próximo tópico.

Operadores Lógicos da Linguagem Object Pascal

A linguagem Object Pascal tem os seguintes operadores lógicos:

Operador	Significado
not	Negação
and	"e" lógico
or	"ou" lógico
xor	"ou" lógico exclusivo

A tabela a seguir mostra o resultado de expressões em que são usados os operadores lógicos.

Operando A	Operando B	not A	A and B	A or B	A xor B
True	True	False	True	True	False
True	False	False	False	True	True
False	True	True	False	True	True
False	False	True	False	False	False

Fundamentos em: Funções e Procedimentos

PRÉ-REQUISITOS

- Noções básicas da utilização dos ambientes de desenvolvimento do Delphi 7.

METODOLOGIA

- Apresentação e descrição dos conceitos e elementos de sintaxe necessários à criação dos procedimentos e funções.

TÉCNICA

- Apresentação e descrição dos procedimentos necessários à criação de procedimentos e funções na linguagem Object Pascal.

A Estratégia de Dividir Para Conquistar

O conceito de procedimentos e funções advém da necessidade de se subdividir um sistema complexo em unidades menores, denominadas sub-rotinas (denominação genérica dada aos procedimentos e funções).

Essa estratégia, popularmente conhecida como "dividir para conquistar", facilita a manutenção de um sistema e a reutilização de código em diversos aplicativos.

Nos próximos tópicos serão apresentados os procedimentos necessários à definição dos procedimentos e funções na linguagem Object Pascal.

Procedimentos (Procedures)

Normalmente, os procedimentos são usados para dividir um programa em blocos menores de código e para armazenar trechos de código utilizados diversas vezes no programa (em vez de se digitar o mesmo trecho de código em cada ponto do programa no qual a sua presença se faz necessária, basta que se inclua uma chamada ao procedimento).

Normalmente, o cabeçalho de um procedimento é composto pela palavra reservada *procedure*, seguida do nome do procedimento, uma lista de parâmetros e um ponto-e-vírgula.

A definição de um procedimento na linguagem Object Pascal obedece à seguinte sintaxe:

```
procedure nome_do_procedimento (parâmetro_1: tipo_1, ,...,parâmetro_n: tipo_n)
var
{declaração de variáveis locais ao procedimento}
begin
{Corpo do procedimento}
end;
```

A palavra reservada *var* indica o início do trecho de código em que são declaradas as variáveis locais ao procedimento. A declaração de variáveis locais termina na linha anterior à da palavra reservada *begin*, que inicia o corpo do procedimento propriamente dito, e que termina com a palavra reservada *end*, seguida de um ponto-e-vírgula.

Um exemplo de procedure muito utilizada no Delphi 7, quando se quer exibir uma mensagem simples para o usuário, é a procedure ShowMessage, cuja declaração é mostrada a seguir.

```
procedure ShowMessage(const Msg: string);
```

Essa procedure tem como único parâmetro uma string a ser exibida em uma caixa de diálogo. Para exibir a mensagem "Alô pessoal", basta incluir a seguinte linha de código:

```
Showmessage('Delphi 7');
```

A mensagem será exibida em uma caixa de diálogo simples, como mostra a figura a seguir.

Figura 3.2: Exibindo uma mensagem com a procedure ShowMessage.

O quadro de diálogo gerado pela procedure Showmessage exibe, na sua barra de títulos, o nome do aplicativo, além de um botão com o texto "OK", usado para fechar a caixa de diálogo. Esse exemplo foi elaborado com um novo projeto, inicialmente denominado Project1.

Funções

Na linguagem Object Pascal, uma função é muito semelhante a um procedimento, com a diferença de que a chamada a uma função deve retornar um valor como resultado, e este pode ser atribuído a uma variável. Além disso, o resultado de uma chamada a função pode ser diretamente incorporado a uma expressão aritmética.

Normalmente, o cabeçalho de uma função é composto pela palavra reservada *function*, seguida pelo nome da função, uma lista de parâmetros, um sinal de dois-pontos, do tipo de retorno e um ponto-e-vírgula.

A definição de uma função na linguagem Object Pascal obedece à seguinte sintaxe:

```
function nome_da_função (parâmetro_1:tipo_1;...;parâmetro_n:tipo_n): tipo_de_retorno;
var
{declaração de variáveis locais à função}
begin
   {Corpo da função}
   result:= valor;
end;
```

A palavra reservada *var* indica o início do trecho de código em que são declaradas as variáveis locais à função. A declaração de variáveis termina na linha anterior à da palavra reservada *begin*. A palavra reservada *begin* inicia o corpo da função propriamente dita, que termina com a palavra reservada *end*, seguida de um ponto-e-vírgula.

O valor a ser retornado deve ser atribuído a result (uma variável interna criada automaticamente pelo ambiente) ou ao próprio nome da função, como indicado.

Um grupo de funções bastante útil no Delphi 7 é o das funções que permitem a conversão de tipos.

Por exemplo, para converter um número real em uma string, deve ser utilizada a função FloatToStr, que tem o seguinte cabeçalho:

```
function FloatToStr(Value: Extended): string;
```

Como você pode observar, essa função recebe um parâmetro do tipo Extended e retorna uma string.

Para converter uma string em um número real, deve ser utilizada a função StrToFloat, que tem o seguinte cabeçalho:

```
function StrToFloat(const S: string): Extended;
```

Nesse caso, a função recebe um parâmetro do tipo string e retorna um número real.

Tanto para procedimentos como para funções, a cláusula var só deve ser incluída se você realmente declarar uma variável local à função. Caso não se declare nenhuma variável local, a cláusula var deve ser omitida, ou dará origem a um erro de compilação.

Funções e Procedimentos Para Manipulação de Arquivos Representados por Variáveis

A relação a seguir apresenta as principais funções para manipulação de arquivos representados por uma variável:

- Append (var F): Abre o arquivo representado pela variável F, apenas para escrita no final do arquivo.
- AssignFile(var F; FileName: string): Associa à variável F o arquivo cujo nome é passado como segundo parâmetro.
- CloseFile (var F): Fecha o arquivo representado pela variável F.
- EOF (var F): Retorna True, se o arquivo representado pela variável F está posicionado no seu final, e False, em caso contrário.
- Erase (var F): Apaga o arquivo representado pela variável F.
- FileSize (var F): Retorna o tamanho, em bytes, do arquivo representado pela variável F.
- Read (F, V1 [, V2, ..., Vn]): Lê elementos de dados em um arquivo representado pela variável F e os armazena nas variáveis v1, v2, ..., vn.
- Readln ([var F: Text;] V1 [, V2, ..., Vn]): Lê elementos de dados em uma linha de um arquivo de texto representado pela variável F e os armazena nas variáveis v1, v2, ..., vn. Caso não sejam fornecidos parâmetros, o arquivo passa para a linha seguinte.
- Rename (var F; NewName): Renomeia como NewName o arquivo representado pela variável F.
- Reset (var F [: File; RecSize: Word]): Esse procedimento abre o arquivo representado pela variável F. O parâmetro RecSize é opcional e especifica o tamanho do registro usado na transferência de dados. Se for omitido, o valor default 128 é usado. Se o arquivo não existir, ocorrerá um erro no processamento. Se o arquivo já estiver aberto, ele é fechado e reaberto, sendo posicionado no seu início. Se F representar um arquivo de texto, ele é aberto apenas para leitura.
- Rewrite (var F [: File; RecSize: Word]): Esse procedimento cria o arquivo representado pela variável. Se o arquivo já existir, seu conteúdo será apagado, mesmo que já esteja aberto.
- Write (F, V1 [, V2,..., Vn]): Escreve, em um arquivo representado pela variável F, elementos de dados armazenados nas variáveis v1, v2, ..., vn.
- Writeln ([var F: Text;] V1 [, V2, ..., Vn]): Escreve, em uma linha de um arquivo de texto representado pela variável F, elementos de dados armazenados nas variáveis v1, v2, ..., vn.

Caso não sejam fornecidos parâmetros, o arquivo escreve uma linha em branco e passa para a linha seguinte.

Funções e Procedimentos Para Manipulação Direta de Arquivos

A relação a seguir apresenta as principais funções para manipulação direta de arquivos (não associados a uma variável):

- ChangeFileExt (const FileName, Extension: string): Muda para Extension a extensão do arquivo cujo nome e/ou path completo são definidos pela string FileName.
- DeleteFile (const FileName: string): Apaga o arquivo cujo nome e/ou path completo são definidos pela string FileName. Retorna False, se o arquivo não existe, e True, em caso contrário.
- ExpandFileName (const FileName: string): Retorna em uma string o path completo e o nome do arquivo definido pela string FileName.
- ExtractFileDir (const FileName: string): Retorna em uma string o diretório do arquivo cujo nome e/ou path completo são definidos pela string FileName.
- ExtractFileDrive (const FileName: string): Retorna em uma string o drive do arquivo cujo nome e/ou path completo são definidos pela string FileName.
- ExtractFileExt (const FileName: string): Retorna em uma string a extensão do arquivo cujo nome e/ou path completo são definidos pela string FileName.
- ExtractFileName (const FileName: string): Retorna em uma string apenas o nome do arquivo cujo nome e/ou path completo são definidos pela string FileName.
- ExtractFilePath (const FileName: string): Retorna em uma string apenas o path completo do arquivo cujo nome e/ou path completo são definidos pela string FileName.
- FileExists (const FileName: string): Retorna True, se o arquivo cujo nome e/ou path completo são definidos pela string FileName existe, e False, em caso contrário.
- FileSearch (const Name, DirList: string): Pesquisa, pelos diretórios definidos no parâmetro DirList, um arquivo cujo nome é definido pela string Name. O parâmetro DirList é uma string em que os diretórios de pesquisa devem ser separados por vírgulas. Se o arquivo for encontrado, a função retorna o path completo do arquivo.
- RenameFile (const OldName, NewName: string): Renomeia para NewName o arquivo cujo nome é definido pela string OldName, retornando True, se a operação é realizada com sucesso, e False, em caso contrário.

Fundamentos em: Classes e Objetos

PRÉ-REQUISITOS

- Noções básicas da utilização dos ambientes de desenvolvimento do Delphi 7.

METODOLOGIA

- Apresentação e descrição dos conceitos de classes e objetos.

TÉCNICA

- Apresentação e descrição dos procedimentos necessários à criação de classes e objetos na linguagem Object Pascal.

Uma Nova (Mas Já Não Tão Nova) Abordagem

As técnicas de programação orientada a objetos são a última palavra em programação, mudando a forma de concepção de um sistema.

A programação orientada a objetos permite que um sistema seja subdividido em entidades denominadas objetos, capazes de representar um sistema de uma forma muito semelhante àquela pela qual encaramos o mundo ao nosso redor.

Como essa tecnologia se baseia no conceito de classes e objetos, os próximos tópicos apresentam os procedimentos necessários à sua implementação na linguagem Object Pascal.

As Classes e os Objetos

A linguagem Object Pascal é uma linguagem de programação orientada a objetos. A expressão POO (Programação Orientada a Objetos) tem tirado o sono de muitos programadores iniciantes, principalmente quando estes resolvem aprender programação orientada a objetos em textos que abordam o tema sob um aspecto bastante filosófico (e pouco didático).

Como este livro se destina a ensinar ao leitor como desenvolver aplicativos utilizando o Delphi 7 como ferramenta de desenvolvimento, e como foge aos nossos objetivos escrever um tratado sobre POO, nos próximos tópicos procuraremos abordar de maneira didática e sucinta apenas os conceitos básicos da POO necessários ao entendimento dos exemplos apresentados ao longo desta primeira parte do livro.

Na segunda parte do livro, abordaremos mais detalhadamente as técnicas de programação orientada a objetos.

O Conceito de Classes

Nos tópicos anteriores, mostramos como criar uma variável composta em Object Pascal. Como você deve se lembrar, uma variável composta tem diversos campos, que podem ser de qualquer tipo predefinido da linguagem ou previamente criados pelo programador.

Naquele tópico, vimos que, para criar um tipo de variável composta denominada materia, com os campos Livro_Texto e carga_horaria, bastava incluir o seguinte trecho de código na seção type da unit:

```
materia= Record
     Livro_Texto: string;
     carga_horaria: integer;
end;
```

Vimos também que, para declarar uma variável denominada matematica, do tipo materia, bastava incluir a sua declaração após a palavra reservada *var*:

```
var
    matematica: materia;
```

Uma classe, por sua vez, tem muitas semelhanças com uma variável composta, pois uma classe também pode ter diversos campos. A definição de um tipo de classe pode ser feita de forma bastante semelhante à de um tipo de variável composta, bastando que se substitua a palavra reservada *Record* por *Class*.

Dessa maneira, se quiséssemos criar uma classe denominada materia em vez de um tipo composto chamado materia, bastaria incluir o seguinte trecho de código após a palavra reservada *type*:

```
materia= Class
     Livro_Texto: string;
     carga_horaria: integer;
end;
```

Agora teríamos uma classe denominada materia, em vez de um tipo de variável composta chamada materia.

Podemos, então, declarar um objeto denominado matematica, da classe materia, incluindo a sua declaração após a palavra reservada *var*:

```
var
    matematica: materia;
```

Repare que, até o presente momento, pouca coisa mudou. A diferença básica é que anteriormente criávamos um tipo de variável e declarávamos uma variável daquele tipo. Agora criamos uma determinada classe e declaramos um objeto daquela classe.

Podemos, então, concluir que uma classe está para um tipo de variável assim como um objeto de uma determinada classe está para uma variável de determinado tipo! (os puristas da POO dizem que um objeto é uma instância de uma classe).

Para acessar os campos de um objeto da classe, basta que se utilize a notação de ponto, descrita anteriormente para as variáveis compostas.

Para atribuir valores aos campos do objeto matematica (da classe materia), basta incluir as seguintes linhas de código:

```
matematica.Livro_Texto:= 'Matematica para Principiantes';
matematica.carga_horaria:= 75;
```

Alguns puristas da POO preferem dizer que um objeto é uma instância de uma classe. Na minha opinião, o importante é que você entenda o conceito de classes e objetos, razão pela qual não estou me atendo rigorosamente aos termos técnicos da Programação Orientada a Objetos.

Métodos de uma Classe

No tópico anterior, vimos as semelhanças existentes entre tipos de variáveis compostas e classes, e transformamos a variável composta materia em uma classe.

Por enquanto você não deve ter percebido qualquer vantagem na utilização de classes, pois tudo o que fizemos com classes até o momento poderia ser feito com uma variável composta.

Neste tópico, com a apresentação do conceito de criação de métodos, essas vantagens começarão a aparecer.

Suponha que se queira criar uma função capaz de obter a carga horária semanal de uma matéria, uma vez conhecida a sua carga horária total.

Poderíamos, a princípio, criar uma função que recebesse como argumentos a carga horária total da matéria e o número de semanas de aula, retornando como resposta o valor da carga horária semanal.

Essa função poderia ser definida como mostra o trecho de código a seguir:

```
function carga_semanal (disciplina: materia; semanas: integer): integer;
begin
   result:= (disciplina.carga_horaria div semanas);
end;
```

A função anterior deve ser implementada na seção Implementation de uma unit, sendo apenas o seu cabeçalho reproduzido na função interface, pelas razões expostas no próximo parágrafo.

Caso se queira que outras unidades de código (units) que incluem o nome da unit na qual essa função foi definida em sua cláusula uses possam utilizar essa função, o cabeçalho da função deve ser definido na seção Interface da sua unit.

Essa função precisa de dois parâmetros: um objeto da classe matéria e um número inteiro, que representa o número de semanas de um período letivo. Como resultado, a função retorna a carga horária semanal, obtida dividindo-se a carga horária total (que é um campo do objeto) pelo número de semanas do período letivo.

Repare que a função foi definida externamente à classe e que um objeto da classe é um dos parâmetros da função.

Que tal se essa função fizesse parte da nossa classe? É aqui que começam as diferenças entre uma classe e uma variável composta.

Para que a função anterior faça parte da classe, basta defini-la (ou declará-la) na própria definição da classe, como mostra o trecho de código a seguir:

```
type
materia= Class
     Livro_Texto: string;
     carga_horaria: integer;
     function carga_semanal (disciplina: materia; semanas: integer): integer;
end;
```

A implementação da função, no entanto, deve ser feita fora da classe, na seção Implementation da unit, como mostra o trecho de código a seguir (correspondente a uma unit do Delphi 7):

```
implementation
{$R *.DFM}
```

```
function materia.carga_semanal (disciplina: materia; semanas: integer): integer;
begin
      result:= (disciplina.carga_horaria div semanas);
end;
```

Repare, contudo, que agora o nome da função é precedido pelo nome da classe, indicando que essa função pertence à classe, isto é, a função é um *método* da classe.

Mas as coisas podem ficar ainda melhores! Como a função é agora um método da classe, ela enxerga os campos da classe. Dessa maneira, não há mais a necessidade de se passar um objeto da classe como parâmetro da função, e esse método pode ser redefinido como mostrado nos trechos de código a seguir:

```
type
materia= Class
     Livro_Texto: string;
     carga_horaria: integer;
     function carga_semanal (semanas: integer): integer;
  end;
.....................................
implementation
{$R *.DFM}
f
unction materia.carga_semanal (semanas: integer): integer;
begin
      result:= (carga_horaria div semanas);
end;
```

Aí está mais uma vantagem de se empregar um método! Ao contrário das funções externas à classe, um método conhece e pode acessar todos os campos da classe.

Para executar um método de uma classe, basta chamar o nome do método usando-se a mesma notação de ponto utilizada para acessar os campos da classe (devendo, no entanto, incluir os parâmetros necessários à chamada do método), como mostra o trecho de código a seguir:

```
var
   matematica: materia;
   horas_por_semana: integer;

.....

horas_por_semana:= matematica.carga_semanal(15);
```

Métodos sem Parâmetros

Ao longo do livro, você verá algumas chamadas a métodos que não têm parâmetros. Isso ocorre quando todos os valores a serem manipulados pelo método são campos da classe, e não há necessidade de se passar qualquer parâmetro para o método.

No exemplo descrito nos tópicos anteriores, se horas_por_semana e semanas fossem campos da classe, esse método não precisaria de parâmetros, como mostra o trecho de código a seguir:

```
type
materia= Class
     Livro_Texto: string;
     carga_horaria, horas_por_semana, semanas: integer;
```

```
      function carga_semanal: integer;
  end;

.....

implementation
{$R *.DFM}
function materia.carga_semanal: integer;
begin
      horas_por_semana:= (carga_horaria div semanas);
end;
```

Nesse caso, para executar o método, basta que se inclua a seguinte linha de código (neste caso, nenhum parâmetro é necessário na chamada do método):

```
matematica.carga_semanal;
```

Onde, como descrito nos tópicos anteriores, matematica é um objeto da classe materia.

Entretanto, nos casos em que uma função não precisa retornar um valor, é mais conveniente que se defina o método como uma procedure em vez de uma função.

Dessa maneira, poderíamos redefinir o método como mostra o trecho de código a seguir:

```
type
materia= Class
     Livro_Texto: string;
     carga_horaria, horas_por_semana, semanas: integer;
     procedure carga_semanal;
  end;
.....................................
implementation

{$R *.DFM}

procedure materia.carga_semanal: integer;
begin
      horas_por_semana:= (carga_horaria div semanas);
end;
```

Embora nesse caso o uso de uma função não gere um erro de compilação, a utilização de uma procedure evita que o Delphi 7 exiba a mensagem de advertência "Return value of function materia.carga_horaria might be undefined".

O OBJETO FORMULÁRIO E A CLASSE TFORM

Examinando novamente o código gerado pelo Delphi 7, notamos a seguinte declaração de classe no arquivo UnitPrincipal.pas:

```
type
TForm1 = class(TForm)
    private
        { Private declarations }
    public
        { Public declarations }
  end;
```

Fazendo uma analogia com a definição da classe materia analisada nos tópicos anteriores, observa-se que esse trecho de código corresponde à declaração de uma nova classe, chamada TForm1.

A diferença é que, após a palavra reservada *class*, aparece entre parênteses a palavra *TForm*. Mas o que significa isso? Isso tem a ver com o conceito de herança, também muito importante na programação orientada a objetos. Esse conceito, bastante importante, será discutido nos próximos tópicos, junto com o significado das palavras reservadas *private* e *public*.

Bom, por enquanto vamos nos concentrar apenas no fato de que TForm1 é uma nova classe. Se continuarmos a examinar o código gerado pelo Delphi 7, veremos ainda a seguinte declaração, na seção var da unit:

```
var
   Form1: TForm1;
```

Como você já deve ter percebido, essa declaração indica que Form1 é um *objeto* da classe TForm1.

Esse objeto é nada mais nada menos que o formulário criado automaticamente pelo Delphi 7, e do qual você já tomou conhecimento desde o início do segundo capítulo.

Mas o que é TForm? TForm é uma classe, definida na unit Forms (no caso da VCL) ou Qforms (no caso do CLX), e que representa um formulário vazio (sem nenhum controle ou componente).

Então você deve estar se perguntando: não seria mais fácil trabalhar com a classe TForm, em vez de criar uma nova classe, chamada TForm1?

Bem, a resposta seria positiva se você quisesse trabalhar com um formulário vazio. Acontece que esse, provavelmente, não é o seu caso; afinal de contas, para que serve um formulário (janela) vazio?

No próximo tópico abordaremos o conceito de herança de classes, e acredito que as coisas começarão a ficar mais claras.

O Conceito de Herança de Classes

Um dos conceitos mais importantes da programação orientada a objetos é o de herança de classes. Por meio da herança, podemos criar uma nova classe baseada em uma classe já existente. Quando uma classe é derivada de uma já existente, diz-se que essa nova classe herda os campos e métodos de uma classe-base.

Dessa maneira, a forma mais genérica de declaração de uma classe obedece à sintaxe abaixo:

```
nome_da_classe = class(nome_da_classe_base)
private
    { Campos e métodos privados}
    public
    { Campos e métodos public
  end;
```

Mas você deve estar se perguntando: qual é a classe-base da classe materia, definida nos tópicos anteriores?

A resposta é a seguinte: a linguagem Object Pascal possui uma classe, denominada TObject, que é a mãe de todas as classes. Por esta razão, quando você deriva uma classe diretamente de TObject, não

precisa declarar explicitamente a classe-base, pois o ambiente do Delphi 7 assume a classe TObject como a classe-base default.

Desse modo, as declarações de classe a seguir são equivalentes:

```
materia= Class
     Livro_Texto: string;
     carga_horaria, horas_por_semana, semanas: integer;
     procedure carga_semanal;
  end;
```

Ou:

```
materia= Class(TObject)
     Livro_Texto: string;
     carga_horaria, horas_por_semana, semanas: integer;
     procedure carga_semanal;
  end;
```

Podemos então concluir que, ao se criar uma nova classe sem especificar uma classe-base, estamos na realidade criando uma classe derivada de TObject.

No caso anterior, o Delphi 7 criou uma nova classe chamada TForm1, derivada de TForm. A razão de se criar uma nova classe é que, como será visto posteriormente, à medida que inserirmos componentes e controles em nosso formulário, esses objetos passarão a ser um campo da nova classe (um campo pode ser qualquer tipo já definido, inclusive um objeto de uma outra classe).

Nos tópicos a seguir, serão apresentados os tipos de métodos e campos de uma classe.

Tipos de Métodos e Campos

Na linguagem Object Pascal, uma classe pode ter os seguintes tipos de métodos e campos:

- Públicos (public).
- Privados (private).
- Protegidos (protected).

Além dos campos dos tipos citados anteriormente, existem ainda os tipos published e automated. O tipo published é semelhante ao tipo public, mas seu valor pode ser visualizado no Object Inspector (e será visto em maiores detalhes no capítulo referente à criação de componentes). O tipo automated também tem as mesmas regras de visibilidade do tipo public e geralmente é usado em classes derivadas da classe TAutoObject (definida na unit OleAuto).

Métodos e Campos Públicos

Os métodos e campos públicos de uma classe são definidos após a palavra reservada *public* e podem ser acessados em qualquer ponto de um programa.

Na linguagem Object Pascal, os campos de uma classe são public por default (isto é, se nada for especificado, o campo será considerado public).

Métodos e Campos Privados

Os métodos e campos privados de uma classe são definidos após a palavra reservada *private* e só podem ser acessados na unit em que a classe foi definida.

Métodos e Campos Protegidos

Os métodos e campos protegidos de uma classe são definidos após a palavra reservada *protected* e só podem ser acessados na unit em que a classe foi definida ou, em outra unit, pelas classes dela derivadas.

Ao longo do livro, ao lidarmos com controles e componentes, usaremos eventualmente o termo *propriedade* quando nos referirmos a um campo de um objeto, seja esse campo visível ou não no Object Inspector.

Procedimentos Associados a Eventos

Se você observou atentamente o procedimento associado ao evento OnActivate de um formulário, mostrado no capítulo anterior, viu que o Delphi 7 cuidou de quase tudo.

Mesmo assim, é bom que você entenda o significado de cada trecho de código em um procedimento, para se sentir mais à vontade ao escrever o código do seu aplicativo.

Vamos examinar atentamente o código do procedimento, que é novamente exibido a seguir.

```
procedure TForm1.FormActivate(Sender: TObject);
begin

end;
```

Inicialmente, temos na primeira linha o cabeçalho do procedimento, que começa com a palavra reservada *procedure*, seguida do nome da classe do objeto (TForm1), de um ponto e do indicativo do evento (FormActivate – Ativação de formulário), além de uma lista de parâmetros (Sender: TObject) e de um ponto-e-vírgula, que indica o fim do cabeçalho do procedimento.

Abaixo do cabeçalho vem o corpo do procedimento, limitado pelas palavras reservadas *begin* e *end*, entre as quais deve ser digitado o código do procedimento (nesse caso, o código que definirá o comportamento do nosso aplicativo em resposta à ocorrência do evento).

Além disso, se você observar o início do arquivo de código, verá que o Delphi 7 também incluiu na definição do objeto TForm1 a declaração da procedure, como mostra o trecho de código a seguir, extraído do arquivo de código.

```
type
  TForm1 = class(TForm)
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Pode-se concluir, portanto, que o procedimento associado ao evento OnActivate, denominado FormActivate é, nesse caso, um método da classe TForm1.

Essa procedure tem apenas o parâmetro Sender, que identifica o objeto que gerou o evento e, exceto em situações em que vários objetos compartilham o mesmo evento, não precisa ser diretamente manipulado pelo programador.

Pare e Reflita Antes de Prosseguir

Uma última recomendação: antes de passar diretamente aos próximos capítulos, verifique se realmente compreendeu os fundamentos da linguagem Object Pascal.

Lembre-se de que ambientes de desenvolvimento como o do Delphi 7 facilitam muito o seu trabalho, principalmente no que se refere à criação da interface, mas não desenvolve o sistema para você. Cabe a você, desenvolvedor, codificar a aplicação de forma a resolver um elenco de questões.

Não se iluda! Programar não é o mesmo que desenhar uma belíssima interface. O trabalho de criação de interface, embora extremamente importante, não representa o coração de um sistema.

No próximo capítulo falaremos sobre a importância de um bom planejamento – requisito indispensável ao sucesso de um empreendimento. Leia-o com atenção, pois os aspectos discutidos poderão lhe ser muito úteis.

Capítulo 4

Planejando a sua Aplicação

Neste capítulo será mostrada a importância de se planejar uma aplicação antes de iniciar o seu desenvolvimento.

Fundamentos em: Planejamento de Aplicações

PRÉ-REQUISITOS

- Disciplina de trabalho e capacidade organizacional.

METODOLOGIA

- Apresentação e descrição dos fatores que influenciam o sucesso no desenvolvimento de uma aplicação.

Planejando o seu Trabalho

Um bom planejamento é indispensável ao sucesso de um empreendimento. Nos próximos tópicos, serão descritos alguns aspectos julgados importantes ao planejamento do trabalho de um desenvolvedor.

A Importância de um Bom Planejamento

Antes de iniciar a codificação e o desenho da interface da sua aplicação, é importante que você reserve um tempo para o seu planejamento.

Esse planejamento, embora possa parecer supérfluo para algumas pessoas, pode reduzir bastante o tempo despendido no desenvolvimento da sua aplicação.

Graças à facilidade e rapidez com que se pode construir a interface de uma aplicação com o Delphi 7, muitos programadores começam a desenhar a interface sem se preocupar com um planejamento prévio da sua aplicação e, no meio do processo de desenvolvimento, descobrem que muita coisa poderia ser modificada (e, nesse caso, as correções podem consumir um tempo muito maior do que aquele gasto no seu planejamento).

A fim de evitar esse tipo de problema, é recomendável que se reserve algum tempo para discutir aspectos importantes da aplicação, como, por exemplo:

- Que menus devem ser incluídos na tela principal da aplicação? E quais devem ser os itens de cada menu?
- O programa terá uma barra de ferramentas? E uma barra de status? Que botões devem ser incluídos na barra de ferramentas? Que informações devem ser exibidas na barra de status?
- Como será o pano de fundo da tela principal da aplicação? Será utilizada uma cor padrão ou um bitmap?
- O programa efetuará acesso a tabelas de bancos de dados? Em caso afirmativo, qual o tipo de banco de dados a ser empregado (Access, Paradox, MySQL, dBASE, Interbase, Oracle, etc.)? Haverá algum relacionamento entre essas tabelas? Desktop ou Client/Server? Será uma aplicação multiplataforma (desenvolvida com a CLX)?

- Que relatórios serão gerados pela aplicação?
- Como deve ser o help on-line da aplicação? Que itens devem ser incluídos e como esses itens devem estar relacionados?
- Quantas janelas e caixas de diálogo deve ter a aplicação? Qual a função de cada uma delas?
- Como será o instalador do aplicativo? Será utilizado o InstallShield Express, um outro produto comercial ou será desenvolvido um instalador para a aplicação?

Como você deve ter percebido, o elenco de questões é muito grande (e olhe que relacionamos apenas poucos itens). Se você planejar direitinho o seu trabalho, o desenvolvimento da aplicação será mais rápido e menos sujeito a erros.

No caso do desenvolvimento em equipe, esse planejamento é fundamental, pois a divisão de tarefas só deve ser feita após uma perfeita definição dos seus objetivos específicos.

Planejando o Nosso Aplicativo-exemplo

O aplicativo-exemplo que será desenvolvido ao longo desta primeira parte do livro será destinado a gerenciar o cadastro de sócios de um clube fictício. Os conceitos de programação apresentados, no entanto, se aplicam a outros tipos de aplicações, pois as técnicas apresentadas na construção da interface e para acesso a bancos de dados permanecem as mesmas.

Nossa aplicação deverá permitir:

- O cadastro de novos sócios.
- A alteração de dados de um sócio.
- A exclusão de um sócio.
- A consulta de dados dos sócios.
- O cadastro de novas atividades.
- A alteração de dados de uma atividade.
- A exclusão de uma atividade.
- A consulta de dados das atividades.
- O cadastro de novas matrículas em atividades.
- A exclusão de matrículas em atividades.
- A consulta de dados das matrículas em atividades.

A definição dos objetivos acima é o primeiro passo a ser dado no planejamento da nossa aplicação.

Com base nesses objetivos, podem-se definir:

- Os menus necessários à nossa aplicação.

- As tabelas necessárias ao armazenamento dos dados: tabelas com os dados dos sócios, das atividades e das matrículas.
- As janelas que devem compor a aplicação: uma janela para cadastro de sócios, outra para cadastro de atividades e janelas para matrículas em atividades, etc… Além disso, deve ser prevista uma caixa de diálogo com informações sobre direitos autorais do programa.
- Os relatórios que devem ser gerados pela aplicação.
- Os itens a serem incluídos no nosso arquivo de help.

Esses itens serão detalhados nos próximos capítulos, na medida em que forem apresentadas as técnicas utilizadas na sua elaboração.

Padronizando a Nomenclatura dos Componentes

Ao criar a interface da sua aplicação, você incluirá diversos componentes nos vários formulários que a compõem. Cada formulário, controle ou componente terá um nome (definido na sua propriedade Name) pelo qual será referenciado no código da aplicação.

Quando você inicia uma nova aplicação (um novo projeto), o Delphi 7 cria automaticamente um formulário denominado Form1. Se você criar um segundo formulário, ele será denominado Form2 e assim por diante.

Imagine agora que a sua aplicação possua quinze formulários (o que não é tanta coisa assim). Já imaginou ter de se lembrar qual a função de Form1, Form2, ..., Form15?

Quando se inserem componentes em um formulário, ocorre a mesma coisa. Se você colocar quatro caixas de texto em um formulário, a primeira será denominada Edit1, a segunda Edit2 e assim por diante.

Para facilitar as suas tarefas como desenvolvedor de aplicações, você deve estabelecer uma convenção para os nomes dos seus formulários e componentes. Pode ser qualquer uma, desde que seja de fácil entendimento.

Alguns autores recomendam a utilização de prefixos (que indicam o tipo do componente) seguidos de um nome que identifique claramente a que se destina o componente.

Eu, por exemplo, costumo chamar de FormPrincipal o formulário principal de uma aplicação. Com relação a componentes do tipo Label, usados apenas para exibir textos estáticos, só altero seu nome quando preciso modificar alguma das suas propriedades no código do aplicativo. Uma caixa de texto na qual o usuário deve digitar seu nome costumo denominar EditNome, por exemplo.

Mas, conforme já disse anteriormente, não há uma regra rígida para os nomes dos componentes. A melhor regra é aquela que mais facilita o seu trabalho. O importante é que você defina uma e a utilize de forma coerente. No caso em que o desenvolvimento é feito em equipe, a utilização de uma convenção é ainda mais importante.

Fundamentos em: To-Do Lists

PRÉ-REQUISITOS

- Experiência prévia no desenvolvimento de aplicações com Delphi, e na manipulação do seu ambiente de desenvolvimento.

METODOLOGIA

- Apresentação do problema: Utilização das TO-DO Lists na organização e planejamento durante o desenvolvimento de aplicações com o Delphi 7.

TÉCNICA

- Apresentação dos procedimentos necessários à utilização do recurso de TO-DO Lists.

Organizando o seu Trabalho com as Ferramentas To-Do List

Desenvolver um bom aplicativo requer, antes de mais nada, um bom planejamento e uma prévia organização das tarefas a serem desempenhadas, e é justamente esta a função das To-Do Lists, presentes no Delphi 7.

Sua função consiste em registrar atividades que devem ser realizadas no desenvolvimento de um projeto de aplicativo. Uma referência a uma atividade pode ser adicionada a um projeto na própria janela que relaciona as To-Do Lists ou diretamente no código-fonte da aplicação.

A figura a seguir apresenta a janela de gerenciamento das To-Do Lists de um projeto, exibida quando se seleciona o item To-Do List do menu View do ambiente de desenvolvimento integrado do Delphi 7:

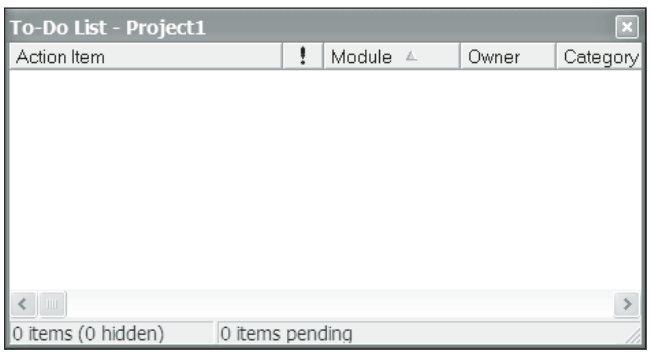


Figura 4.1: A janela de gerenciamento das To-Do Lists.

Cada Projeto possui a sua própria relação de To-Do Lists a ele associada.

Adicionando um Item a uma To-Do Lists

Para adicionar um item a uma To-Do Lists, você deve executar os seguintes procedimentos:

1. Exibir a janela de To-Do Lists do projeto.
2. Selecionar o botão direito do mouse sobre esta janela e, no menu pop-up que será exibido, selecionar o item Add, para exibir a caixa de diálogo Add To-Do Item, mostrada na figura a seguir.

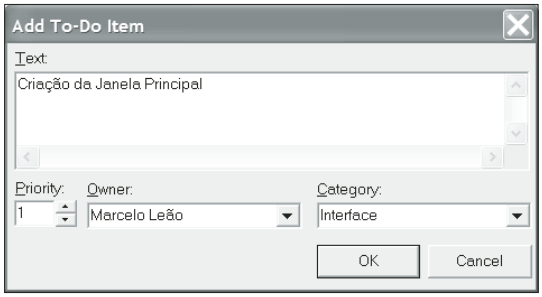


Figura 4.2: A caixa de diálogo Add To-Do Item.

Nesta caixa de diálogo, deverão ser fornecidas as seguintes informações:

- Um texto descritivo da ação a ser realizada (Text).
- A prioridade desta tarefa (Priority).
- O responsável pela tarefa (Owner). Você pode digitar um novo nome ou selecionar um nome já existente.
- A sua categoria (Category). Você pode digitar uma nova categoria ou selecionar um nome já existente.

3. Após fornecer as informações necessárias, selecionar o botão Ok para fechar esta caixa de diálogo e criar o novo item.

Editando um Item de uma To-Do Lists

Para editar um item de uma To-Do Lists, você deve executar os seguintes procedimentos:

1. Exibir a janela de To-Do Lists do projeto.
2. Selecionar o botão direito do mouse sobre um item desta janela e, no menu pop-up que será exibido, selecionar o item Edit, para exibir a caixa de diálogo Edit To-Do item, mostrada na figura a seguir.

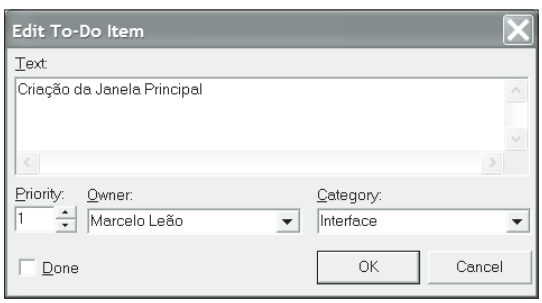


Figura 4.3: A caixa de diálogo Edit To-Do Item.

Nesta caixa de diálogo, poderão ser alteradas as informações já fornecidas na criação do item, além de poder indicar que a tarefa já foi realizada (o que é feito marcando-se a caixa de verificação Done).

3. Após fornecer as informações necessárias, selecionar o botão Ok para fechar esta caixa de diálogo.

Excluindo um Item de uma To-Do Lists

Para excluir um item de uma To-Do Lists, você deve executar os seguintes procedimentos:

1. Exibir a janela de To-Do Lists do projeto.
2. Selecionar o item a ser removido.
3. Pressionar o botão Del ou selecionar o botão direito do mouse sobre esta janela e, no menu pop-up que será exibido, selecionar o item Delete.

Configurando as Informações Exibidas em uma To-Do Lists

Para configurar as informações a serem exibidas em uma TO-DO Lists, você deve executar os seguintes procedimentos:

1. Exibir a janela de TO-DO Lists do projeto.
2. Selecionar o botão direito do mouse sobre esta janela e, no menu pop-up que será exibido, selecionar o item Table Properties, para exibir a caixa de diálogo mostrada na figura a seguir.

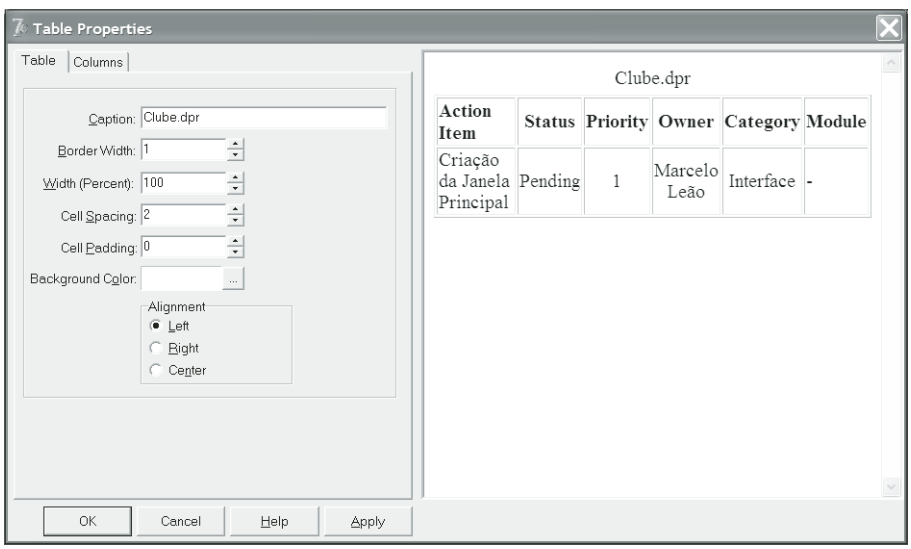


Figura 4.4: A caixa de diálogo Table Properties.

Esta caixa de diálogo apresenta duas páginas, uma referente às características gerais da tabela de itens, como mostrado na figura anterior, e outra na qual se pode configurar cada uma das suas colunas, como mostrado na próxima figura.

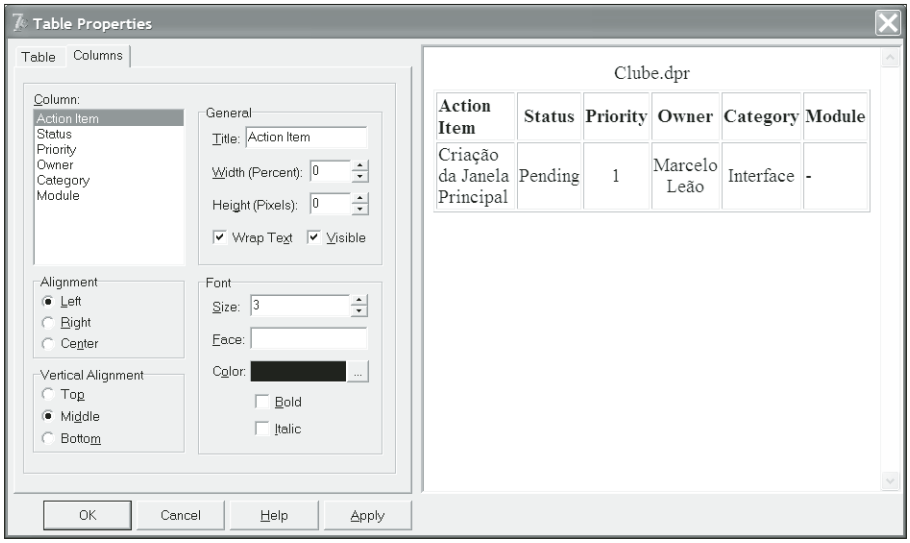


Figura 4.5: A página Column da caixa de diálogo Table Properties.

Nesta página pode-se definir, para cada coluna:

- Seu título (Title).
- Sua largura (Width) em porcentagem da largura total.
- Sua altura (Height) em pixels.
- Se o texto pode ser quebrado em mais de uma linha (Wrap).
- Se a coluna deve estar visível (Visible).
- Seu alinhamento (horizontal e vertical).
- A fonte usada na exibição do texto correspondente.

Configurando Como as Informações Devem Ser Ordenadas em uma To-Do Lists

Para configurar como as informações devem ser ordenadas em uma To-Do Lists, você deve executar os seguintes procedimentos:

1. Exibir a janela de To-Do Lists do projeto.
2. Selecionar o botão direito do mouse sobre esta janela e, no menu pop-up que será exibido, selecionar o item Sort, para exibir os subitens pelos quais a lista pode ser ordenada, selecionando-se então o item desejado.

Adicionando um Item a uma To-Do Lists Diretamente no Código-fonte

Para adicionar um item a uma To-Do Lists diretamente no código-fonte, você deve executar os seguintes procedimentos:

1. No Editor de código, selecione o botão direito do mouse e, no menu suspenso que será exibido, selecione o item Add To-Do item. Será exibida a caixa de diálogo Add To-Do item, na qual deverá ser selecionado o item desejado e o botão Ok, para inserir a referência a este item no código-fonte, que deverá ficar como apresentado a seguir (o código-fonte se refere ao procedimento associado ao evento OnClick de um botão chamado BotaoTeste, criado apenas para ilustrar o uso das TO-DO Lists):

```
procedure TFormPrincipal.BotaoTesteClick(Sender: TObject);
begin
  { TODO 2 -oMarcelo -cInterface : Iniciar a interface }
end;
```

Repare que será criado um novo item na TO-DO List – Neste caso seria melhor não tê-lo criado anteriormente. A diferença está no fato de que o item recém-criado está diretamente vinculado ao código-fonte e, ao ser posteriormente editado e marcado como concluído, o código-fonte será automaticamente alterado para:

```
procedure TFormPrincipal.BotaoTesteClick(Sender: TObject);
begin
  { DONE 2 -oMarcelo -cInterface : Iniciar a interface }
end;
```

Você também pode digitar diretamente as informações no código-fonte, usando comandos com a seguinte sintaxe:

```
{TODO|DONE [n] [-o<owner>] [-c<category>] : <to-do item text>}
```

Onde:

- n indica a prioridade do item.
- TODO: indica que a tarefa representada por este item não foi completada.
- DONE: indica que a tarefa representada por este item foi completada.
- -o owner: o nome do responsável pela tarefa.
- -c category: a categoria da tarefa representada pelo item.

Os itens digitados no código-fonte são automaticamente adicionados à To-Do List.

Copiando a Relação de Itens de uma To-Do List

Você pode copiar a relação de itens de uma To-Do List como texto ou no formato HTML.

Para isto, basta selecionar o botão direito do mouse sobre esta janela e, no menu pop-up que será exibido, selecionar o item Copy As para exibir os subitens, selecionando-se então o item desejado.

Filtrando a Relação de Itens de uma To-Do List

Você pode filtrar a relação de itens de uma To-Do List.

Para isto, basta selecionar o botão direito do mouse sobre a janela TO-DO Lists e, no menu pop-up que será exibido, selecionar o item Filter para exibir os subitens, selecionando-se então a opção desejada.

Capítulo 5

Criando o Formulário Principal da Aplicação

Neste capítulo serão apresentados os procedimentos necessários à criação do formulário principal do nosso aplicativo-exemplo. Serão apresentadas as principais propriedades de um formulário, os procedimentos necessários à definição dos seus valores e a inclusão de componentes para a criação da interface.

Fundamentos em: Manipulação de Formulários

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Conhecimentos básicos da codificação na linguagem Object Pascal.
- Noções básicas de programação orientada a objetos, principalmente os conceitos de propriedades, métodos e eventos (apresentados nos capítulos anteriores).

METODOLOGIA

- Apresentação e descrição das principais características de um formulário.

TÉCNICA

- Apresentação dos procedimentos necessários à configuração de formulários.

Os Formulários – Elementos Para a Construção da Interface em Aplicações Desenvolvidas em Delphi 7

Conforme descrito anteriormente, os formulários são as janelas que irão compor a interface da sua aplicação. Este nome pode parecer não muito adequado, mas já está consagrado na literatura, e iremos adotá-lo.

Nos próximos tópicos você aprenderá a definir as principais características do formulário principal de uma aplicação desenvolvida em Delphi 7.

O Objeto Formulário

Um dos principais objetos utilizados no desenvolvimento de aplicações com o Delphi 7 é o objeto formulário (Form).

Como já foi visto no Capítulo 2, sempre que iniciamos uma nova aplicação (um novo projeto), o Delphi 7 cria um formulário vazio, que pode ser usado como o formulário principal da aplicação.

Como todo objeto que se preza, um formulário possui propriedades, métodos e eventos.

Dentre as propriedades de um formulário, podem-se destacar:

- BorderStyle: Determina o estilo de borda do formulário.
- BorderIcons: Determina os ícones a serem habilitados na extremidade superior direita da barra de títulos do formulário.
- Caption: Armazena o texto exibido na barra de títulos do formulário.

- Color: Define a cor do formulário.
- Font: Define a fonte do texto exibido no formulário.
- Height: Define a dimensão vertical (altura) de um formulário.
- Icon: Define o ícone a ser exibido quando o formulário for minimizado.
- Left: Define a posição de um formulário, em relação à extremidade esquerda da tela.
- Menu: Define o menu associado ao formulário.
- Name: Define o nome pelo qual o objeto é referenciado no código da aplicação.
- PopupMenu: Define o menu flutuante associado ao formulário.
- Position: Define o posicionamento do formulário na tela.
- Windowstate: Determina o estado de exibição do formulário (maximizado, minimizado ou normal).
- Top: Define a posição de um formulário, em relação à extremidade superior da tela.

Propriedades com um Conjunto de Valores Predefinidos

Existem propriedades que só podem assumir alguns valores predefinidos. Esse é o caso, por exemplo, da propriedade Position de um formulário, descrita no tópico anterior.

Nesses casos, o valor da propriedade pode ser selecionado a partir de uma lista exibida quando se clica com o botão esquerdo do mouse sobre a seta exibida à direita do valor da propriedade, como mostra a figura a seguir.

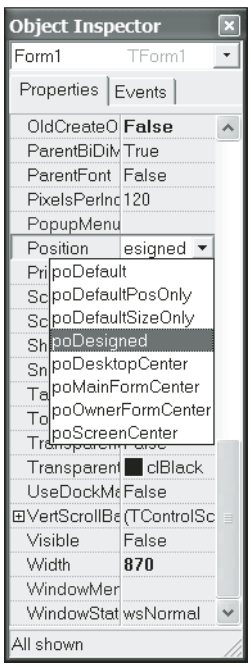


Figura 5.1: Definindo o valor da propriedade Position no Object Inspector.

Propriedades Que Contêm Subpropriedades

Existem algumas propriedades que representam um conjunto de valores. Como foi mostrado no Capítulo 3, um conjunto armazena seus elementos entre colchetes.

Esse é o caso, por exemplo, da propriedade BorderIcons de um formulário.

Uma observação mais atenta mostra que o Object Inspector exibe, à esquerda do nome dessa propriedade, um pequeno sinal de +. Esse sinal indica que essa propriedade é, neste caso, um conjunto, cujos elementos podem ser definidos (habilitados) de forma independente.

Para acessar cada um desses elementos, que denominaremos subpropriedades, basta dar um duplo clique com o botão esquerdo do mouse sobre o sinal +. Nesse caso, cada subpropriedade poderá ser definida de maneira independente, como mostra a figura a seguir. Repare ainda que o sinal + foi substituído por -. Dando-se um duplo clique sobre o sinal -, as subpropriedades ficarão novamente ocultas, e o sinal - será substituído por +.

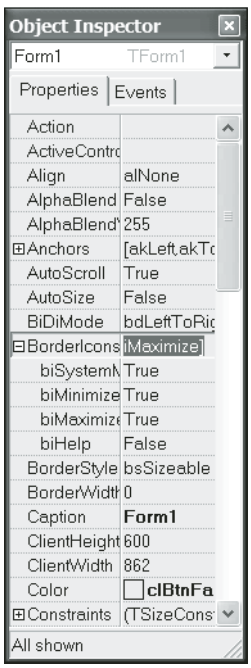


Figura 5.2: Acessando subpropriedades no Object Inspector.

Observe ainda que cada uma dessas propriedades exibe uma lista de valores possíveis (nesse caso, True e False).

Existem ainda casos em que os valores dessas subpropriedades podem ser definidos em uma caixa de diálogo. Esse é o caso, por exemplo, da propriedade Font de um formulário.

Nessas situações, além do sinal de + exibido à esquerda do nome da propriedade, o Object Inspector apresenta reticências (...) à direita do valor da propriedade. Clicando-se sobre essas reticências, exibe-se uma caixa de diálogo na qual se podem definir os valores das subpropriedades.

As figuras a seguir mostram a propriedade Font, e a caixa de diálogo utilizada para atribuir valores às suas subpropriedades.

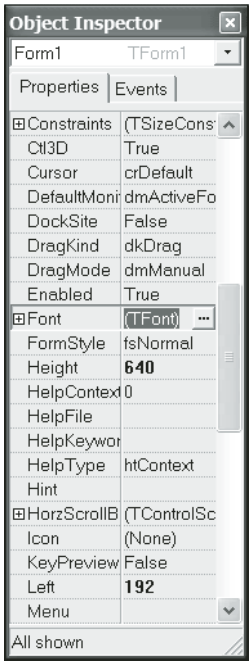


Figura 5.3: Acessando a propriedade Font no Object Inspector.

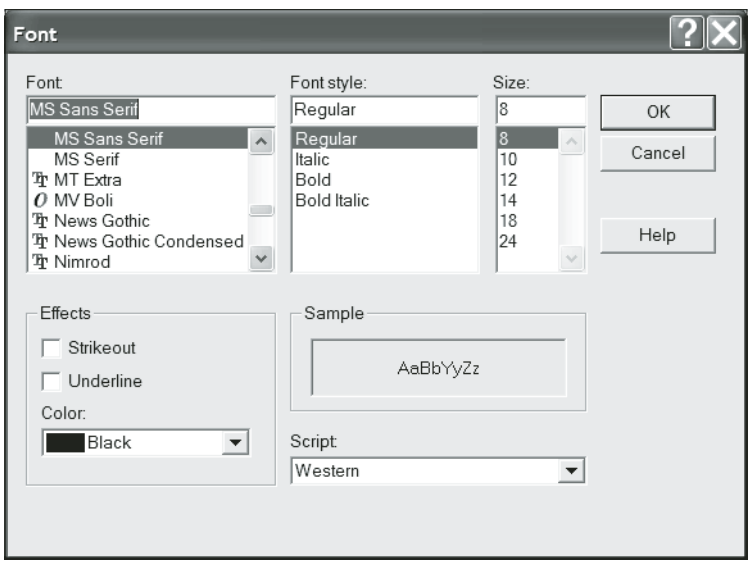


Figura 5.4: A caixa de diálogo Font.

Conforme veremos ao longo do livro, essa situação ocorre em muitas propriedades de diversos componentes.

Alterando as Propriedades do Objeto Formulário

Vamos iniciar a criação da interface visual da nossa aplicação. Para isso, execute os seguintes procedimentos:

1. Abra o projeto Clube.dpr, criado anteriormente.
2. Selecione o formulário principal, inicialmente denominado Form1.
3. Atribua os seguintes valores para as principais propriedades do formulário principal, diretamente no Object Inspector:

   BorderStyle: bsSingle (esse estilo impede que o formulário seja redimensionado).
   BorderIcons: [biSystemMenu,biMinimize].
   Caption: .Cadastro de Sócios.
   Height: 700.
   Name: FormPrincipal.
   Position: poScreenCenter.
   Width: 1100.

As propriedades cujo valor não for alterado permanecerão com o seu valor default.

Os valores das propriedades Height e Width foram definidos considerando-se uma resolução de 1280 por 1024. Caso a resolução a ser adotada seja diferente, reajuste estes valores de forma proporcional.

Seu formulário deve ficar com o aspecto mostrado na figura a seguir.

Figura 5.5: Aspecto inicial do formulário principal da aplicação.

Definindo um Ícone Para o Formulário Principal da Aplicação

Para definir um ícone para o formulário, você deve executar os seguintes procedimentos:

1. Selecione o formulário.
2. Selecione as reticências exibidas à direita da propriedade Icon no Object Inspector. Será exibida a caixa de diálogo Picture Editor, mostrada na figura a seguir.

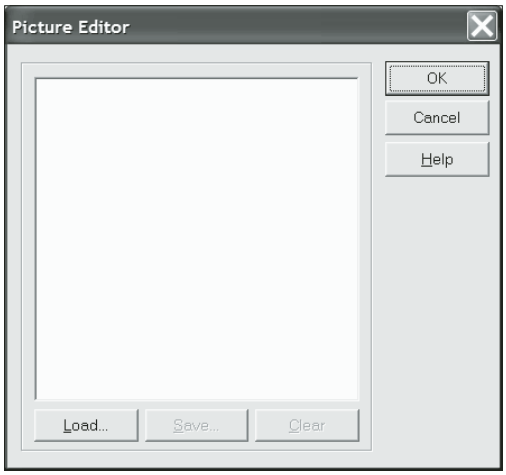


Figura 5.6: A caixa de diálogo Picture Editor.

3. Selecione o botão Load desta caixa de diálogo. Será exibida a caixa de diálogo Load Picture, mostrada na Figura 5.7, na qual deverá ser selecionado o ícone desejado.

Nesse exemplo, a imagem selecionada está no arquivo C:\Arquivos de Programas\Arquivos Comuns\Borland Shared\Images\Icons\Handshak.ico do Windows. A localização deste arquivo pode, no entanto, não ser a mesma no seu computador, dependendo de como foi feita a sua instalação do Delphi 7. Neste caso, recomenda-se pesquisar a sua localização.

4. Selecione o botão Abrir, para fechar a caixa de diálogo Load Picture.

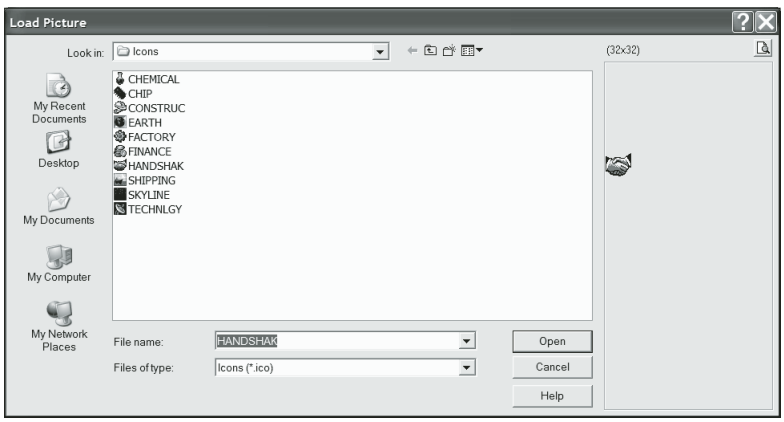


Figura 5.7: A caixa de diálogo Load Picture.

5. Selecione o botão OK, para fechar a caixa de diálogo Picture Editor.

Pronto! Você acaba de definir um ícone para o seu formulário.

Você não precisa distribuir o arquivo de ícone com a aplicação, pois o ícone é incorporado diretamente ao seu executável (você pode verificar isso analisando a descrição textual do formulário).

Inserindo Componentes em um Formulário

Para criar a interface da sua aplicação, você precisará inserir controles e componentes em seus formulários.

Para inserir um componente em um formulário, basta que se execute um dos seguintes procedimentos:

- Dê um duplo clique com o botão esquerdo do mouse sobre o seu ícone na paleta de componentes. Isso faz com que o componente seja exibido centralizado no formulário com o seu tamanho default.

Ou:

- Selecione o ícone correspondente na paleta de componentes e depois dê um clique com o botão esquerdo do mouse sobre o formulário. Isso faz com que o componente seja exibido com o seu tamanho default e com a sua extremidade superior esquerda situada na posição definida pelo ponteiro do mouse.

Ou:

- Selecione o componente desejado na paleta de componentes (clicando uma única vez sobre o seu ícone), pressione o botão esquerdo do mouse sobre o formulário e, mantendo o botão esquerdo do mouse pressionado, arraste-o de forma a definir o tamanho do componente. Isso permite que se definam simultaneamente o tamanho e a posição do componente durante a sua inserção no formulário.

Para inserir mais de um componente de um mesmo tipo em um formulário, pressione a tecla Shift ao selecioná-lo na paleta de componentes. Isso faz com que, ao se clicar novamente sobre o formulário, seja criado um novo componente, sem a necessidade de selecioná-lo outra vez na paleta. Para cancelar o processo de inserção múltipla, basta selecionar qualquer outro componente da paleta.

Observe que os componentes são distribuídos por várias páginas da paleta de componentes (Standard, Additional, Data Access, Data Controls, etc.), e antes de selecionar um componente é necessário que a página onde o mesmo se encontra esteja selecionada (o que é feito clicando-se com o botão esquerdo do mouse sobre o nome da página).

Inserindo um Componente Para Exibição de Imagens no Formulário Principal

Para incluir um componente capaz de exibir imagens gravadas em arquivos, execute os seguintes procedimentos:

1. Selecione a página Additional da paleta de componentes.

2. Selecione o sexto componente da paleta.
3. Clique com o botão esquerdo do mouse sobre o formulário.
4. O componente será exibido no formulário, como mostra a figura a seguir.

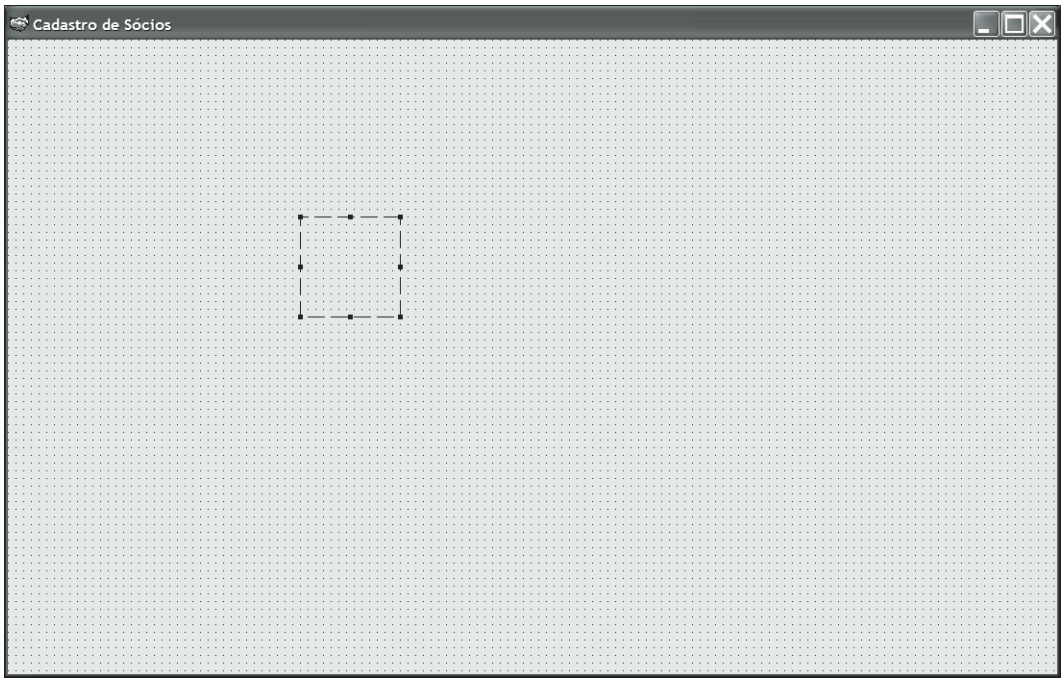


Figura 5.8: Inserindo um componente para exibição de imagens no formulário.

Entre as principais propriedades desse componente, é possível destacar:

- AutoSize: Essa propriedade define se o componente será ou não automaticamente redimensionado para se adaptar ao tamanho da imagem a ser exibida (definida na sua propriedade Picture) e só pode assumir os valores False (falso) e True (verdadeiro).
- Height: Define a dimensão vertical (altura) do componente.
- Left: Define a posição do componente em relação à extremidade esquerda do formulário.
- Name: Define o nome com o qual o objeto é referenciado no código da aplicação.
- Picture: Define a imagem a ser exibida pelo componente.
- Stretch: Essa propriedade define se a imagem será ou não automaticamente redimensionada e, para se adaptar ao tamanho do componente, só pode assumir os valores False (falso) e True (verdadeiro). Entretanto, definir o valor de Stretch como True pode provocar distorções na imagem.
- Top: Define a sua posição, em relação à extremidade superior do formulário.
- Width: Define a dimensão horizontal (largura) do componente.

O valor definido na propriedade AutoSize prevalece sobre o definido na propriedade Stretch.

Selecionando um Componente Inserido em um Formulário

Para selecionar um componente inserido em um formulário, basta clicar sobre este com o botão esquerdo do mouse. Quando um componente é selecionado, apresenta pequenos quadrados pretos ao seu redor. Esses pequenos quadrados pretos são denominados "marcas de seleção" e indicam que o componente está selecionado.

A figura anterior mostra o componente de exibição de imagens selecionado e suas marcas de seleção.

Quando um componente é inserido em um formulário, ele é automaticamente selecionado.

A seleção de um componente permite que ele seja movimentado e redimensionado com o mouse e que as suas propriedades sejam alteradas diretamente no Object Inspector.

Renomeando um Componente

O nome dado a um componente, definido pela sua propriedade Name, é o nome pelo qual este componente será referenciado no código da aplicação.

Para renomear um componente, execute os seguintes procedimentos:

1. Selecione o componente com o mouse.
2. Altere o valor da sua propriedade Name diretamente no Object Inspector.

Executando os passos anteriores, mude o valor da propriedade Name do componente de exibição de imagens para Logotipo.

Reposicionando um Componente

Para movimentar um componente com o mouse, execute os seguintes procedimentos:

1. Pressione o botão esquerdo do mouse sobre o componente e, mantendo o botão pressionado, movimente o mouse, arrastando o componente. À medida que você movimenta o componente, os valores das suas propriedades Left e Top são exibidos junto ao ponteiro do mouse.
2. Quando o componente estiver situado na posição desejada, libere o botão esquerdo do mouse.

Por exemplo: selecione com o mouse o componente inserido no formulário no tópico anterior e verifique o valor das suas propriedades Left e Top no Object Inspector.

Movimente o componente executando os passos descritos anteriormente e, após colocá-lo em uma nova posição, verifique os novos valores das suas propriedades Left e Top.

Como você já deve ter constatado, os valores das propriedades Left e Top são automaticamente atualizados quando você termina de movimentar o componente. Isso ocorre porque as propriedades

Left e Top definem a posição do componente em relação ao canto superior esquerdo do formulário. A propriedade Left mede, em pixels, a distância da extremidade esquerda do componente em relação à extremidade esquerda do formulário. A propriedade Top mede, em pixels, a distância da extremidade superior de um componente em relação à extremidade superior do formulário (no caso de um formulário, essas propriedades são definidas em relação à tela).

A recíproca também é verdadeira, isto é, se você selecionar um componente e alterar os valores das suas propriedades Left e Top no Object Inspector, o componente será deslocado para uma nova posição no formulário.

Reposicione o componente de exibição de imagens inserido no formulário, executando os seguintes procedimentos:

1. Selecione o componente.
2. No Object Inspector, atribua o valor 260 à sua propriedade Top e 469 à sua propriedade Left.

Você também pode reposicionar um componente que esteja selecionado, usando as combinações de tecla Ctrl + uma das teclas de movimentação do teclado (seta para esquerda, seta para direita, etc.).

Redimensionando um Componente

Para redimensionar um componente com o mouse, execute os seguintes procedimentos:

1. Selecione o componente com o botão esquerdo do mouse.
2. Posicione o ponteiro do mouse sobre uma das marcas de seleção do componente, até que ele se torne uma seta dupla.
3. Pressione o botão esquerdo do mouse e movimente o seu ponteiro, redimensionando o componente. À medida que você redimensiona o componente, os valores das suas propriedades Height e Width são exibidos junto ao ponteiro do mouse (esse recurso não estava disponível nas versões anteriores, e simplifica bastante a tarefa de se redimensionar um componente no formulário).
4. Quando o componente estiver com o tamanho desejado, libere o botão esquerdo do mouse.

Selecione, com o mouse, o componente de exibição de imagens que foi inserido no formulário e verifique, no Object Inspector, o valor das suas propriedades Height e Width. Redimensione o componente executando os passos descritos anteriormente e, após redimensioná-lo, verifique os novos valores das suas propriedades Height e Width.

Como você já deve ter constatado, os valores das propriedades Height e Width são automaticamente atualizados quando você termina de redimensionar o componente. Isso ocorre porque as propriedades Height e Width definem respectivamente os valores, em pixels, da altura e largura do componente.

A recíproca também é verdadeira, isto é, se você selecionar um componente e alterar os valores das suas propriedades Height e Width no Object Inspector, o componente será redimensionado.

Redimensione o componente de exibição de imagens inserido no formulário, executando os seguintes procedimentos:

1. Selecione o componente.
2. No Object Inspector, atribua o valor 139 à sua propriedade Height e 153 à sua propriedade Width.

Você também pode redimensionar um componente que esteja selecionado, usando as combinações de tecla Shift + uma das teclas de movimentação do teclado (seta para esquerda, seta para direita, etc.).

Exibindo uma Imagem

Para definir a imagem a ser exibida, o que é feito definindo-se um valor para a propriedade Picture do componente Logotipo, execute os seguintes procedimentos:

1. Selecione o componente Logotipo.
2. No Object Inspector, selecione a sua propriedade Picture.
3. Dê um clique com o botão esquerdo do mouse sobre as reticências (...) exibidas do lado direito da propriedade. Será exibida a caixa de diálogo Picture Editor.
4. Para definir a figura a ser exibida, selecione o botão Load, que exibe a caixa de diálogo Load Picture, descrita anteriormente.
5. Selecione o arquivo C:\Arquivos de Programas\Arquivos Comuns\Borland Shared\Images\splash\16color\Athena.bmp e clique no botão Open.
6. Selecione o botão OK na caixa de diálogo Picture Editor.

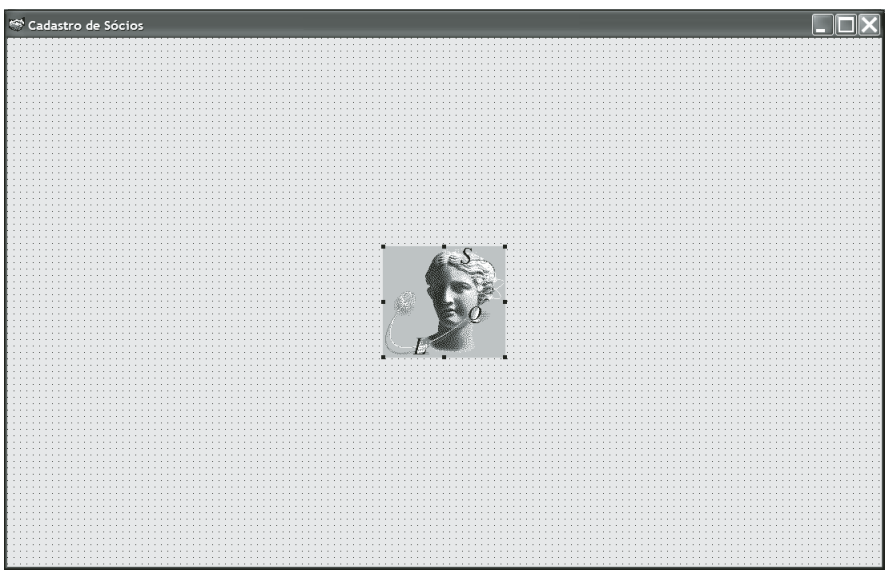


Figura 5.9: Exibição da imagem no componente.

O componente exibirá a figura selecionada (Figura 5.9). No Object Inspector será informado apenas o tipo de imagem a ser exibida pelo componente (TBitmap, TIcon, etc.), e não o diretório e nome do arquivo. Isso se deve ao fato de que a imagem será realmente armazenada no formulário, e não apenas referenciada por ele.

Altere a propriedade AutoSize do componente para True, para que a imagem seja totalmente visível.

Caso você esteja trabalhando com outra resolução, pode centralizar o componente selecionando-o e, com o botão direito do mouse, acessar a opção Position->Align do menu pop-up associado ao componente. Na caixa de diálogo que será exibida, selecione a opção "Center in window" tanto para o alinhamento horizontal como para o alinhamento vertical. Estes procedimentos serão detalhados adiante, no tópico "Alinhando Componentes".

Exibindo um Texto no Formulário

Para exibir em um formulário um texto que não deve ser alterado pelo usuário da aplicação, deve-se usar o componente Label, o quarto componente da página Standard da paleta de componentes.

A principal propriedade desse componente é a propriedade Caption, que define o texto que será exibido.

Para inserir um label no seu formulário, execute os seguintes procedimentos:

1. Selecione a página Standard da paleta de componentes.
2. Selecione o componente Label.
3. Clique com o botão esquerdo do mouse sobre o formulário.
4. Altere o valor da sua propriedade Caption para "Fundamentos do Delphi 7" diretamente no Object Inspector.
5. No Object Inspector, atribua o valor 468 à sua propriedade Left e 550 à sua propriedade Top. O componente será exibido no formulário, como mostra a Figura 5.10.

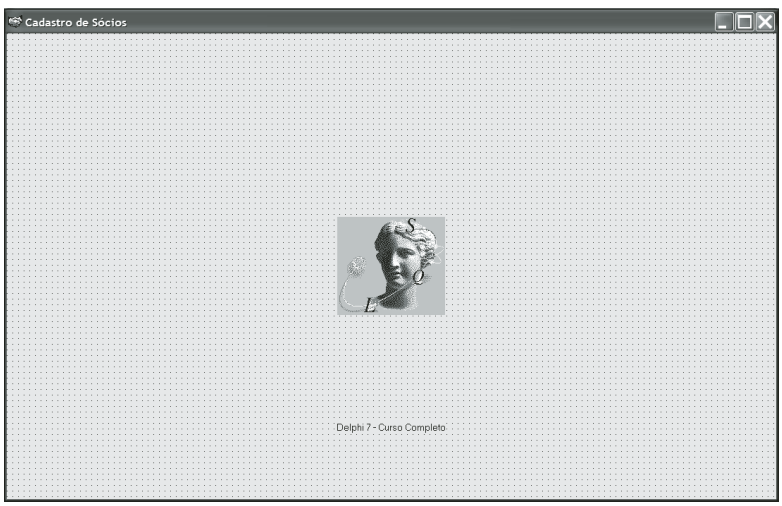


Figura 5.10: Inclusão de um rótulo no formulário.

Alterando a Fonte do Texto Exibido em um Componente

Se você não alterou as configurações padrões do ambiente e seguiu corretamente os passos descritos no tópico anterior, a fonte do texto exibido no rótulo inserido no formulário será a fonte default do ambiente, sem nenhum efeito especial.

Vamos alterar algumas das suas características, de forma a tornar mais atraente o aspecto do formulário principal da nossa aplicação.

Como descrito anteriormente, a propriedade Font de um componente pode ser alterada selecionando este componente e clicando-se com o botão esquerdo do mouse sobre os três pontinhos que são exibidos à direita do valor da propriedade, para exibir a caixa de diálogo Font, mostrada na Figura 5.4. Altere as características da fonte para tamanho 24, estilo Negrito Itálico, efeito sublinhado e cor azul-marinho. Seu formulário deverá ficar com o aspecto mostrado na figura a seguir.

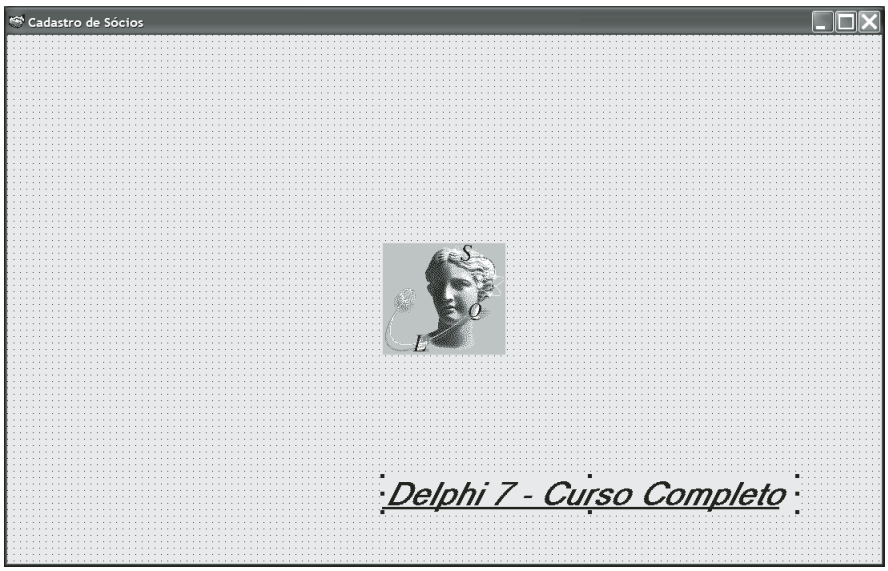


Figura 5.11: Redefinindo a fonte do rótulo no formulário.

Selecionando Vários Componentes Simultaneamente

Para selecionar mais de um componente de uma única vez, execute os seguintes procedimentos:

1. Selecione o primeiro componente com o botão esquerdo do mouse.
2. Pressione a tecla Shift enquanto seleciona os demais componentes.

Você também pode desenhar com o mouse um retângulo que circunscreva todos os componentes a serem selecionados. Para isso, basta executar os seguintes procedimentos:

1. Pressione o botão esquerdo do mouse quando o seu ponteiro estiver sobre o ponto que definirá o primeiro vértice do retângulo no formulário.

2. Mantendo o botão esquerdo do mouse pressionado, arraste o seu ponteiro para a posição que definirá o outro vértice do retângulo (à medida que se arrasta o ponteiro do mouse, um retângulo pontilhado é desenhado automaticamente).
3. Libere o botão esquerdo do mouse.

Após concluir o passo anterior, todos os componentes situados dentro do retângulo ou interceptados por ele estarão selecionados.

Você também pode selecionar todos os componentes presentes em um formulário, selecionando o item Select All do menu Edit do Delphi 7.

Alinhando Componentes

Quando inserimos vários componentes em um formulário, dificilmente conseguimos colocá-los em uma disposição bem organizada. Felizmente, o Delphi fornece um recurso que permite que os componentes sejam facilmente alinhados.

Para ajustar a posição dos componentes Label1 e Logotipo, execute os seguintes procedimentos:

1. Selecione o componente Label1 e, pressionando a tecla Shift, selecione o componente Logotipo. Os dois componentes exibirão marcas de seleção.
2. Selecione o item Align do menu Edit.

Será exibida a caixa de diálogo Alignment, mostrada na Figura 5.12.

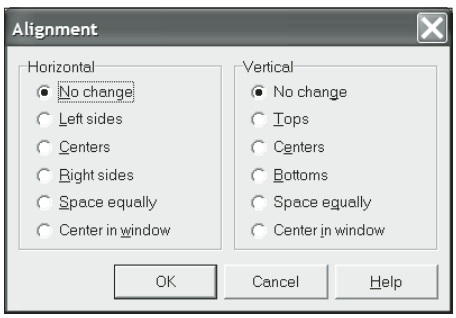


Figura 5.12: A caixa de diálogo Alignment.

Essa caixa de diálogo apresenta dois grupos de botões, um denominado Horizontal e outro denominado Vertical, com as seguintes opções:

- No change: A posição do(s) componente(s) não se altera na direção especificada. Essa opção está disponível nos dois grupos de botões.
- Left sides: Os componentes são alinhados pela sua extremidade esquerda, tomando por base a extremidade esquerda do primeiro componente selecionado. Essa opção só existe no grupo Horizontal.

- Centers: Os componentes são alinhados pelos seus centros, tomando por base o centro do primeiro componente selecionado. Essa opção está disponível nos dois grupos de botões.
- Right sides: Os componentes são alinhados pela sua extremidade direita, tomando por base a extremidade direita do primeiro componente selecionado. Essa opção só existe no grupo Horizontal.
- Space equally: Os componentes são distribuídos de forma que os seus centros se mantenham eqüidistantes. Essa opção está disponível nos dois grupos de botões.
- Center in window: O(s) componente(s) é(são) centralizado(s) na janela. Essa opção está disponível nos dois grupos de botões. Observação: se você selecionar um único componente, este será centralizado na janela, mas, se você selecionar um grupo de componentes, o centro desse grupo é que será alinhado com o centro do formulário.
- Tops: Os componentes são alinhados pela sua extremidade superior, tomando por base a extremidade superior do primeiro componente selecionado. Essa opção só existe no grupo Vertical.
- Bottoms: Os componentes são alinhados pela sua extremidade inferior, tomando por base a extremidade inferior do primeiro componente selecionado. Essa opção só existe no grupo Vertical.

3. Como desejamos alinhar horizontalmente os componentes pelos seus pontos médios, selecione a opção Centers, no grupo Horizontal, e No change, no grupo Vertical.
4. Clique no botão OK para fechar a caixa de diálogo.

Para centralizar horizontalmente os componentes em conjunto no formulário, execute os passos anteriores e selecione a opção Center in Window, nos grupos Horizontal e Vertical da caixa de diálogo Alignment. Seu formulário deve ficar com o aspecto indicado na Figura 5.13.

Figura 5.13: Aspecto do formulário após o alinhamento dos componentes.

Repare que os valores armazenados nas propriedades Left e Top dos componentes devem ter sido alterados.

Analisando o Código Gerado Pelo Delphi

Se você analisar o código da unit correspondente ao formulário, verá a seguinte definição de classe:

```
type
  TFormPrincipal = class(TForm)
  Logotipo: TImage;
  Label1: TLabel;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Repare que a classe TFormPrincipal apresenta, entre seus campos, um componente do tipo TImage e outro do tipo TLabel, que não existem na classe TForm. Lembra-se do conceito de herança? A classe TFormPrincipal, derivada de TForm, herda suas propriedades e métodos, e ainda acrescenta novas propriedades e métodos (inexistentes em TForm).

Testando a sua Aplicação

Até que nosso aplicativo está ficando com uma boa aparência, porém está faltando algo muito comum em todos os aplicativos Windows: um menu. Vamos incluí-lo no próximo capítulo, mas antes vamos fazer um pequeno teste da nossa aplicação.

O Delphi permite que você execute o seu aplicativo a partir de seu próprio ambiente de desenvolvimento. Isso permite que o comportamento e o desempenho da sua aplicação sejam analisados sem a necessidade de sair do ambiente do Borland Delphi 7, otimizando-se o tempo gasto no desenvolvimento da aplicação.

Para testar seu aplicativo, basta executar um dos seguintes procedimentos:

- Selecionar o item Run do menu Run.

Ou:

- Pressionar a tecla de função F9.

Ou:

- Na caixa de ferramentas, selecionar o botão que exibe uma seta verde. Repare que esse botão apresenta uma seta à direita que permite selecionar o projeto a ser executado (no caso de se estar trabalhando com um grupo de projetos).

Ao iniciar a sua execução, o formulário deve apresentar o aspecto da Figura 5.14.

Em princípio, pode parecer que não se tem lá grande coisa, apenas uma janela com um título, uma imagem e nada mais.

Tudo bem, não criamos nenhuma aplicação espetacular, mas sem escrever uma única linha de código já temos uma aplicação que consiste em uma janela principal com uma barra de títulos, botões Maximizar (que nesse caso foi desabilitado por nós), Minimizar, Finalização e um menu de sistema.

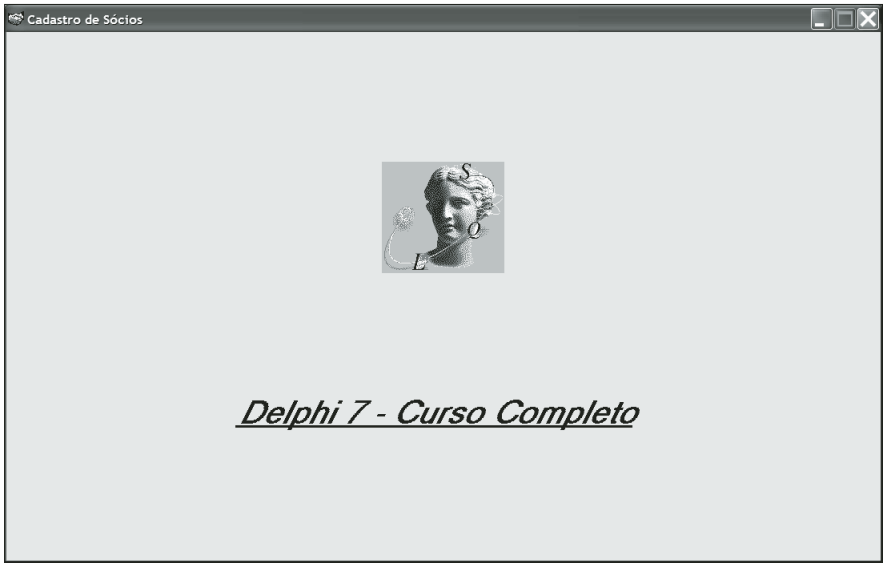


Figura 5.14: Execução do aplicativo a partir do ambiente de desenvolvimento.

Você pode reposicionar a janela, clicando com o botão esquerdo do mouse sobre a sua barra de títulos e, mantendo o botão esquerdo pressionado, arrastá-la para a posição desejada, alternar para outra aplicação e posteriormente retornar. Tudo isso sem a necessidade de qualquer codificação, a não ser aquela gerada pelo próprio Delphi.

De qualquer maneira, isso reforça os argumentos de que a criação da interface é apenas uma fase do desenvolvimento de um aplicativo.

Finalizando a Execução do Aplicativo

Para finalizar o aplicativo, execute um dos seguintes procedimentos:

- Selecione o botão Finalizar (aquele quadradinho com um "x" no seu interior), situado na extremidade direita da barra de títulos.

Ou:

- Selecione o item Fechar no menu de sistema.

Ou:

- Selecione o item Program Reset no menu Run (essa opção só é válida quando você inicializa a aplicação com base no ambiente de desenvolvimento do Borland Delphi).

Ou:

- Pressione simultaneamente as teclas Ctrl e F12 (essa opção só é válida quando você inicializa a aplicação com base no ambiente de desenvolvimento do Borland Delphi).

Capítulo 6

Projetando um Menu Para a sua Aplicação

Neste capítulo, você verá como é simples a inclusão de menus nas janelas formulários da sua aplicação com o Delphi 7, e como associar eventos a itens de um menu.

Fundamentos em: Criação de Menus

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Inclusão e manipulação de componentes em um formulário.

METODOLOGIA

- Apresentação e descrição das principais características de um menu.

TÉCNICA

- Apresentação dos procedimentos necessários à inclusão de um menu no formulário principal da sua aplicação.

Menus – Elementos Indispensáveis ao Formulário Principal de uma Aplicação

Dificilmente encontramos uma aplicação com interface gráfica que não exiba um menu em sua janela principal, e seus aplicativos não devem fugir a essa (quase) regra; afinal de contas seus usuários esperam poder acessar alguns dos recursos dos seus aplicativos por meio dos menus existentes na janela principal.

Nos próximos tópicos mostraremos como é simples incluir essa característica aos aplicativos desenvolvidos com o Delphi 7.

Incluindo um Menu na sua Aplicação

A inclusão de um menu em uma aplicação é feita por meio da inserção de um componente MainMenu no seu formulário principal.

O componente MainMenu está situado na página Standard da paleta de componentes (é o segundo componente desta página) e fornece acesso a um editor de menus.

Para inserir um componente do tipo MainMenu no formulário principal da aplicação, execute os seguintes procedimentos:

1. Exiba o formulário principal.
2. Selecione o componente MainMenu, na página Standard da paleta de componentes, clicando nele com o botão esquerdo do mouse.
3. Clique novamente sobre o formulário, no ponto em que o componente deve ser inserido. O componente MainMenu é inserido no formulário, como mostra a Figura 6.1.
4. Usando o Object Inspector, altere a propriedade Name do componente para MenuPrincipal.

O componente MainMenu é um exemplo de componente não-visual, isto é, não estará visível durante a execução do aplicativo. Ele apenas representa um menu e fornece acesso a um editor de menus; nesse caso, portanto, a sua posição no formulário não é de grande relevância.

Figura 6.1: Inserção de um componente MainMenu no formulário principal da aplicação.

Observe que, quando você inseriu o componente MainMenu no formulário, o ambiente atribuiu o valor MainMenu1 à sua propriedade Name e atribuiu esse mesmo valor à propriedade Menu do formulário (indicando que o menu que será exibido no formulário é o representado por este componente). Quando posteriormente você alterou o valor da propriedade Name do componente MainMenu para MenuPrincipal, o ambiente de desenvolvimento fez a alteração correspondente na propriedade Menu do formulário.

A criação dos menus e itens de menus no Delphi 7 é feita em um editor de menus. No próximo tópico será mostrado como acessar o editor de menus.

Acessando o Editor de Menus

Para acessar o editor de menus, execute os seguintes procedimentos:

1. Selecione o componente MenuPrincipal, criado no tópico anterior.
2. Dê um duplo clique com o botão esquerdo do mouse sobre o componente MainMenu. O editor de menus é exibido, com o primeiro menu selecionado, pronto para ser editado, como mostra a Figura 6.2.

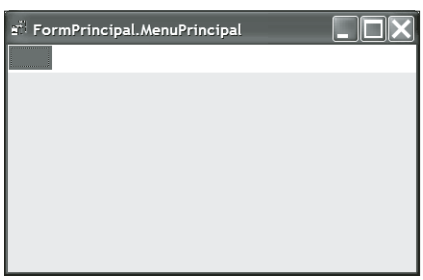


Figura 6.2: O editor de menus.

3. Selecione o Object Inspector e altere o valor da propriedade Name do objeto que está em destaque (um retângulo que representa o primeiro menu) para MenuSistema.
4. Usando o Object Inspector, altere o valor da propriedade Caption para &Sistema.
5. Selecione novamente o editor de menus.

Como você pode ver na Figura 6.3, foi criado o menu Sistema em que a letra S aparece sublinhada. Se você já está acostumado a usar aplicativos Windows, sabe que uma letra sublinhada em um menu funciona como uma tecla de atalho, que permite que o menu seja acessado pressionando-se simultaneamente as teclas Alt e a tecla correspondente à letra sublinhada. Foram criados também dois novos espaços: um espaço à direita, que permite a criação de um novo menu, e um espaço sob o menu Sistema, para que se criem itens desse menu.

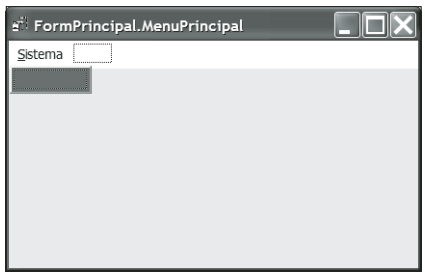


Figura 6.3: Criação do menu Sistema.

Os itens de menu (que são objetos da classe TMenuItem) possuem uma propriedade chamada AutoHotKeys que, se configurada corretamente, impede que dois itens de menu possuam a mesma tecla de atalho.

Esta propriedade pode assumir um dos valores a seguir:

- maAutomatic: Caso dois itens de menu possuam a mesma tecla de atalho, a do último item é automaticamente redefinida durante a execução do aplicativo.
- maManual: Caso dois itens de menu possuam a mesma tecla de atalho, a do último item é redefinida mediante uma chamada ao método RethinkHotKeys da classe TMenuItem.
- maParent: A redefinição das teclas de atallho de itens de menu segue a regra estabelecida pelo seu componente-pai. No caso de itens de menu, por exemplo, será adotado o valor configurado na propriedade de mesmo nome do componente MainMenu ou PopupMenu que o contém.

Você também pode acessar o editor de menus clicando com o botão direito do mouse sobre o componente MenuPrincipal e selecionando o item Menu Designer no menu pop-up exibido. Outra opção consiste em selecionar no Object Inspector a propriedade Items do componente MainMenu.

Crie ainda os seguintes menus para o nosso aplicativo-exemplo:

- Menu Sócios

  Name: MenuSocios
  Caption: &Sócios

- Menu Atividades

  Name: MenuAtividades
  Caption: &Atividades

- Menu Matrícula

  Name: MenuMatricula
  Caption: &Matrícula

- Menu Relatórios

  Name: MenuRelatorios
  Caption: &Relatórios

- Menu Help

  Name: MenuHelp
  Caption: &Help

Repare que a propriedade Name não pode conter acentuação, o que já não ocorre com a propriedade Caption.

A figura a seguir apresenta o aspecto do formulário principal da aplicação, após a inclusão destes menus.

Figura 6.4: Aspecto do formulário principal da aplicação, após a inclusão de todos os menus.

Criando Itens de Menu

Como vimos no tópico anterior, a criação de menus e itens de menu é feita usando-se o editor de menus do ambiente de desenvolvimento.

Um item de menu é, na realidade, um objeto da classe TMenuItem, cujas principais propriedades de um item de menu são:

- Caption: Uma cadeia de caracteres que define o texto exibido pelo item de menu.
- Checked: Uma variável booleana que define se o item de menu deve ou não exibir uma marca de verificação, indicando que alguma opção está ativa. No Microsoft Word, por exemplo, uma marca de verificação no item Régua do menu Exibir indica que a régua está sendo exibida.
- Enabled: Uma variável booleana que define se o item de menu está ou não habilitado. Quando essa propriedade tem o valor False, o item de menu está desabilitado e apresenta um aspecto acinzentado.
- GroupIndex: Essa propriedade define como os menus das janelas-filhas são mesclados com os menus da janela principal em uma aplicação MDI.
- ImageIndex: Define o índice correspondente à imagem do componente Imagelist associado ao componente MenuPrincipal que será exibido com o item de menu. Caso não se queira exibir nenhuma imagem com o menu, o valor dessa propriedade deve ser igual a -1.
- Name: Define o nome pelo qual o objeto é referenciado no código da aplicação (lembre-se de que no Delphi um item de menu é um objeto da classe TMenuItem).
- RadioItem: Uma variável booleana que define se a marca de verificação de um item de menu deve ser uma marca circular.
- ShortCut: Essa propriedade define a combinação de teclas que executa a mesma ação que um item de menu. Essa combinação de teclas é denominada tecla aceleradora de um item de menu.

Em nossa aplicação, devemos oferecer ao usuário a possibilidade de incluir novos sócios, atividades e matrículas, excluí-los ou alterar os seus dados cadastrais, e a cada uma dessas opções pode corresponder um item de menu. Além disso, deve ser capaz de fazer cópia (backup) e restauração destes dados.

Para criar um item Backup no menu Sistema, execute os seguintes procedimentos:

1. Selecione o espaço em branco que foi criado sob o menu Sistema.
2. Selecione o Object Inspector e altere o valor da sua propriedade Name para SistemaBackup.
3. Usando o Object Inspector, altere o valor da propriedade Caption para &Backup.
4. Selecione novamente o editor de menus.

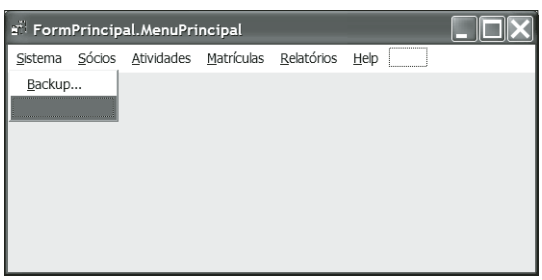


Figura 6.5: Criação do item Backup no menu Sistema.

Como você pode ver na Figura 6.5, foi criado o item Backup no menu Sistema. Também foi incluído um espaço sob o item, para que seja criado um novo item de menu.

Para criar os itens Restaurar e Finalizar deste menu, você deve proceder de forma semelhante, definindo as seguintes propriedades para cada um dos itens.

Item	Name	Caption
Restaurar	SistemaRestaurar	&Restaurar...
Finalizar	SistemaFinalizar	&Finalizar...

A Figura 6.6 mostra o estado atual do nosso menu, após a criação de mais esses dois itens.

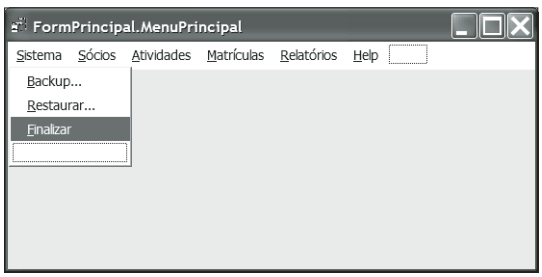


Figura 6.6: Estado atual do menu Sistema.

Criando um Separador de Itens em um Menu

Para criar um separador de itens entre os itens Restaurar e Finalizar do menu Sistema, execute os seguintes procedimentos:

1. Selecione o item Finalizar do menu Sistema.
2. Pressione a tecla Ins para criar um novo item entre os itens Restaurar e Finalizar. Selecione o Object Inspector e altere o valor da propriedade Name deste novo item para Separador.
3. Usando o Object Inspector, altere o valor da propriedade Caption para "-" (isso mesmo, um sinal de subtração, usado para criar um separador entre itens de menu).

Pronto! Está criado o separador de itens de menu, como pode ser verificado na figura a seguir.

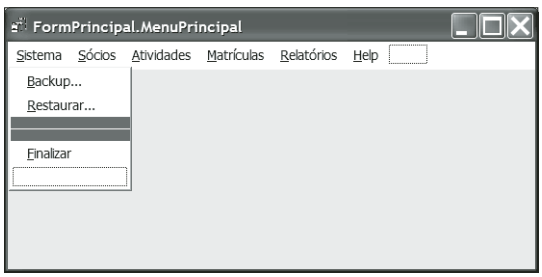


Figura 6.7: Criando um separador de itens de menu.

Os itens de menu (que são objetos da classe TMenuItem) possuem uma propriedade chamada AutoLineReduction que, se configurada corretamente, impede a inclusão de dois separadores contíguos, ou de um separador isolado no final de um menu.

Esta propriedade pode assumir um dos valores a seguir:

- maAutomatic: Caso haja separadores contíguos, os excedentes serão automaticamente removidos durante a execução do aplicativo.
- maManual: Caso haja separadores contíguos, os excedentes serão removidos durante a execução do aplicativo mediante uma chamada ao método RethinkLines da classe TMenuItem.
- maParent: A remoção de separadores contíguos segue a regra estabelecida pelo seu componente-pai. No caso de itens de menu, por exemplo, será adotado o valor configurado para o componente MainMenu ou PopupMenu que o contém.

Criando Teclas Aceleradoras Para Itens de Menu

Para criar teclas aceleradoras para itens de menus, basta definir corretamente sua propriedade ShortCut no Object Inspector.

Desta maneira, para definir como Ctrl+X as teclas aceleradoras do item Finalizar do menu sistema, execute os seguintes procedimentos:

1. Acesse o editor de menus.
2. Selecione o item Finalizar deste menu.
3. No Object Inspector, altere o valor da propriedade ShortCut deste item para Ctrl+X.

A figura a seguir apresenta o aspecto deste menu, após a criação das teclas aceleradoras para o item Finalizar do menu Sistema.

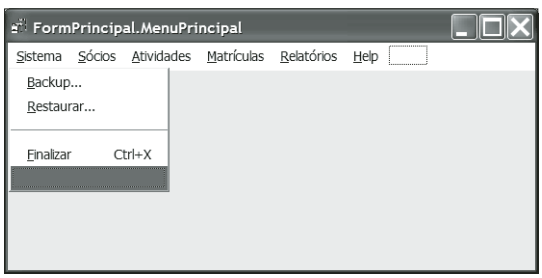


Figura 6.8: Criação das teclas aceleradoras para o item Finalizar do menu Sistema.

Criando Outros Itens de Menu

Devemos ainda criar os itens dos demais menus do nosso aplicativo-exemplo, seguindo os procedimentos que serão apresentados ainda neste tópico.

Para criar um item Cadastrar no menu Sócios, execute os seguintes procedimentos:

1. Acesse o editor de menus e selecione o espaço em branco que foi criado sob o menu Sócios.
2. Selecione o Object Inspector e altere o valor da sua propriedade Name para SociosCadastro.
3. Usando o Object Inspector, altere o valor da propriedade Caption para &Cadastro.
4. Selecione novamente o editor de menus.

Para criar os itens Alteração,Exclusão e Consulta deste menu, você deve proceder de forma semelhante, definindo as seguintes propriedades para cada um dos itens.

Item	Name	Caption
Alteração	SociosAlteracao	&Alteração...
Exclusão	SocioExclusao	&Exclusão
Consulta	SocioConsulta	&Consulta

É uma convenção de que, em programas desenvolvidos para o ambiente Windows, itens de menu que provoquem a exibição de caixas de diálogo terrminem com três pontinhos.

Execute procedimentos semelhantes para criar os itens Cadastro, Alteração, Exclusão e Consulta do menu Atividades.

- Para o menu Matrículas, crie os itens Cadastro, Exclusão e Consulta.
- Para o menu Relatórios, crie os itens Sócios, Atividades e Matrículas.
- Para o menu Help, crie os itens Tópicos e Sobre.

Incluindo um Menu Pop-up na sua Aplicação

A inclusão de um menu pop-up em uma aplicação é feita por meio da inserção de um componente PopupMenu no seu formulário principal (o componente PopupMenu está situado na página Standard da paleta de componentes e é o terceiro componente desta página).

Para inserir um componente do tipo PopupMenu no formulário principal da aplicação, execute os seguintes procedimentos:

1. Selecione o formulário principal.
2. Selecione o componente PopupMenu, na página Standard da paleta de componentes.
3. Clique novamente com o mouse sobre o formulário, no ponto em que o componente deve ser inserido. O componente PopupMenu é inserido no formulário, como mostra a Figura 6.9.
4. Usando o Object Inspector, altere a propriedade Name do componente para PopupPrincipal.

Figura 6.9: Inserção de um componente PopupMenu no formulário principal da aplicação.

5. Usando o Object Inspector, altere a propriedade PopupMenu do componente FormPrincipal para PopupPrincipal.

Criando Itens de Menu em um Menu Pop-up

A criação de itens em um menu pop-up é feita por meio de um editor de menus vinculado a este componente, acessado da mesma forma que no caso do menu principal da aplicação.

Para criar itens do menu pop-up, execute os seguintes procedimentos:

1. Selecione o componente PopupPrincipal, criado no tópico anterior.
2. Dê um duplo clique com o botão esquerdo do mouse sobre o componente PopupMenu. O editor de menus pop-up é exibido, com o primeiro item selecionado, pronto para ser editado, como mostra a Figura 6.10.

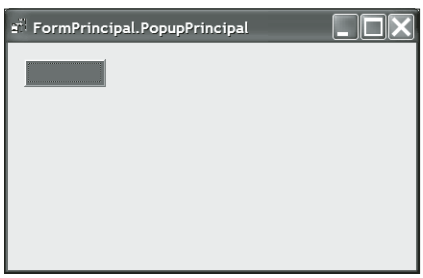


Figura 6.10: Acessando o editor de menus com base no menu pop-up.

3. Altere os valores das suas propriedades Name e Caption para PopupSistema e &Sistema, respectivamente.

4. Pressione o botão de seta para baixo, para exibir o espaço destinado ao próximo menu pop-up, como mostra a Figura 6.11.

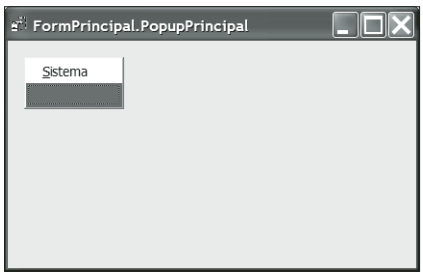


Figura 6.11: Criando o menu Sistema no menu pop-up.

5. Pressione o botão de seta para baixo, para exibir o espaço destinado ao próximo menu pop-up.
6. Altere os valores de suas propriedades Name e Caption para PopupSócios e &Sócios, respectivamente.
7. Pressione o botão de seta para baixo, para exibir o espaço destinado ao próximo menu pop-up.
8. Altere os valores de suas propriedades Name e Caption para PopupAtividades e &Atividades, respectivamente.
9. Pressione o botão de seta para baixo, para exibir o espaço destinado ao próximo menu pop-up.
10. Altere os valores de suas propriedades Name e Caption para PopupMatriculas e &Matrículas, respectivamente.
11. Pressione o botão de seta para baixo, para exibir o espaço destinado ao próximo menu pop-up.
12. Altere os valores de suas propriedades Name e Caption para PopupRelatorios e &Relatórios, respectivamente.
13. Pressione o botão de seta para baixo, para exibir o espaço destinado ao próximo menu pop-up.
14. Altere os valores de suas propriedades Name e Caption para PopupHelp e &Help, respectivamente.

Seu menu pop-up ficará com o aspecto da Figura 6.12.

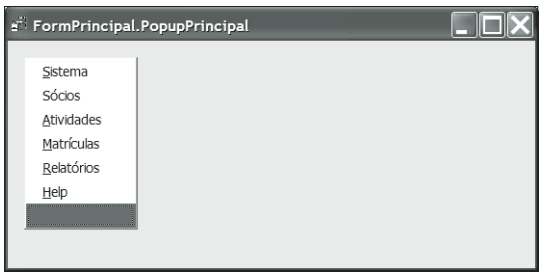


Figura 6.12: Criando o menu pop-up.

Criando Submenus no Menu Pop-up

No tópico anterior criamos os menus pop-up da nossa aplicação. Para incluir os seus itens, no entanto, teremos que introduzir o conceito de submenu.

Para inserir os itens do menu Sistema no submenu correspondente, execute os seguintes procedimentos:

1. Selecione o menu Sistema do menu pop-up e pressione o botão direito, para exibir o menu suspenso mostrado na Figura 6.13.

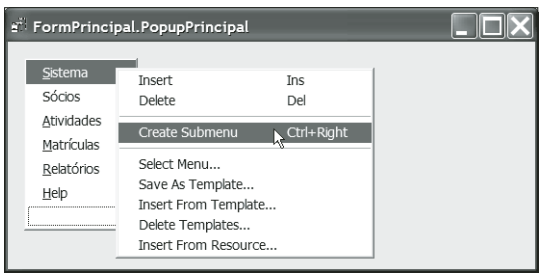


Figura 6.13: Exibindo o menu suspenso do editor de menus.

2. Selecione o item Create Submenu, e o submenu será criado, como mostra a Figura 6.14. A seta exibida ao lado da palavra Sistema foi incluída automaticamente pelo editor de menus.
3. Altere os valores das propriedades Name e Caption desse item para PopupSistemaBackup e &Backup, respectivamente.

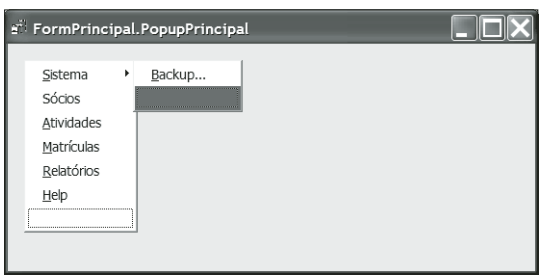


Figura 6.14: Criando um submenu no menu pop-up do editor de menus.

4. Pressione o botão de seta para baixo, para criar um novo item.
5. Altere os valores das suas propriedades Name e Caption para PopupSistemaRestaurar e &Restaurar, respectivamente.
6. Pressione o botão de seta para baixo, para criar um novo item e crie um separador de itens de menu.
7. Pressione o botão de seta para baixo, para criar um novo item.
8. Altere os valores das suas propriedades Name e Caption para PopupSistemaFinalizar e &Finalizar, respectivamente.

Seu menu pop-up deve estar como o da Figura 6.15.

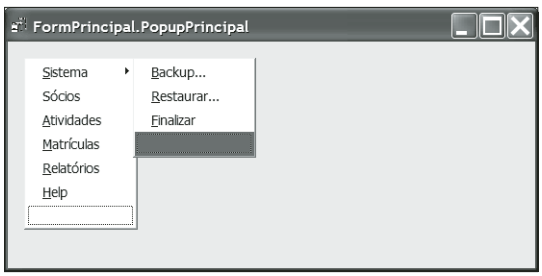


Figura 6.15: Criando itens em um submenu no menu pop-up do editor de menus.

Repita os passos anteriores para criar os demais submenus.

Para o menu pop-up Sócios, inclua os seguintes itens no seu submenu.

Item	Name	Caption
Cadastro	PopupSociosCadastro	&Cadastro...
Alteração	PopupSociosAlteracao	&Alteração...
Exclusão	PopupSocioExclusao	&Exclusão
Consulta	PopupSocioConsulta	&Consulta

Para o menu pop-up Atividades, inclua os seguintes itens no seu submenu.

Item	Name	Caption
Cadastro	PopupAtividadesCadastro	&Cadastro...
Alteração	PopupAtividadesAlteracao	&Alteração...
Exclusão	PopupAtividadesExclusao	&Exclusão
Consulta	PopupAtividadesConsulta	&Consulta

Para o menu pop-up Matriculas, inclua os seguintes itens no seu submenu.

Item	Name	Caption
Cadastro	PopupMatriculasCadastro	&Cadastro...
Exclusão	PopupMatriculasExclusao	&Exclusão
Consulta	PopupMatriculasConsulta	&Consulta

Para o menu pop-up Relatórios, inclua os seguintes itens no seu submenu.

Item	Name	Caption
Sócios	PopupRelatoriosSocios	&Sócios...
Atividades	PopupRelatoriosAtividades	&Atividades
Matrículas	PopupRelatoriosMatriculas	&Matrículas

Para o menu pop-up Help, inclua os seguintes itens no seu submenu.

Item	Name	Caption
Tópicos	PopupHelpTopicos	&Tópicos
Sobre	PopupHelpSobre	&Sobre

ASSOCIANDO EVENTOS A ITENS DE MENU

A associação de um evento a um item de menu é feita de forma semelhante à associação de um evento a qualquer outro componente ou formulário (lembra-se da criação de um procedimento associado ao evento OnActivate de um formulário, feita no Capítulo 2?).

A única diferença é que os itens de menu não são vistos como um componente inserido em um formulário e, para selecionar um deles, você precisa usar a caixa de seleção de objetos do Object Inspector ou o editor de menus.

Vamos agora criar um evento associado ao item Finalizar do menu Sistema. Como o próprio nome indica, queremos que esse item de menu finalize a execução da aplicação; logo devemos incluir a seguinte linha de código no procedimento associado ao evento OnClick desse item de menu:

```
Application.Terminate;
```

Como esse é o formulário principal da aplicação e, ao se fechar o formulário principal de uma aplicação, esta é encerrada, poderíamos alternativamente incluir a seguinte linha de código, em vez da mostrada anteriormente:

```
FormPrincipal.Close;
```

Para criar o procedimento associado ao evento OnClick do item Finalizar do menu Sistema, basta executar os seguintes procedimentos:

1. Na caixa de seleção de objetos do Object Inspector, selecione o objeto SistemaFinalizar, como mostra a Figura 6.16.
2. Selecione a página Events do Object Inspector, se ela já não estiver selecionada.
3. Dê um duplo clique no campo à direita do evento a ser definido (no caso, o evento OnClick).
4. Inclua a linha de código acima, como mostra a Figura 6.17.

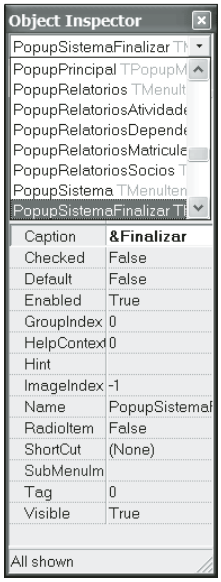


Figura 6.16: Selecionando o objeto SistemaFinalizar no Object Inspector.

Agora sua aplicação pode ser finalizada selecionando-se o item Finalizar do menu Sistema ou por meio da combinação de teclas Ctrl+X. Execute a aplicação e verifique o seu funcionamento.

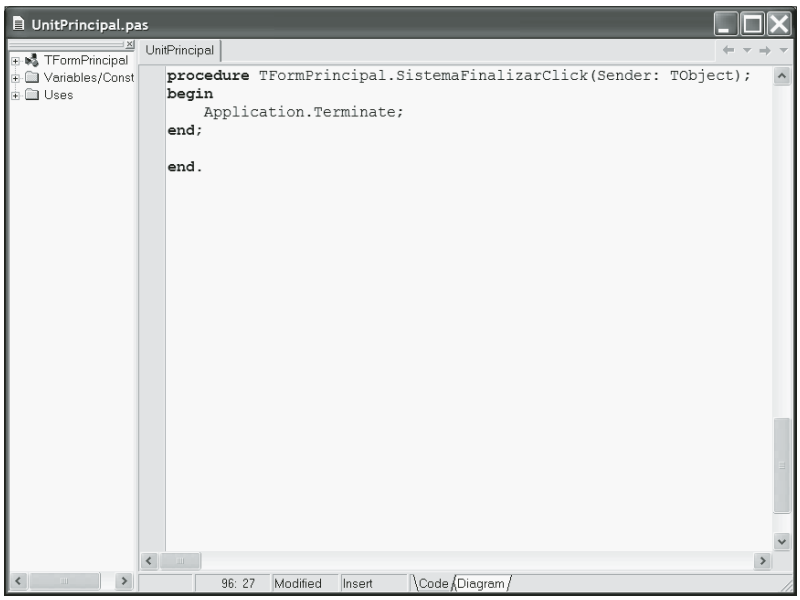


Figura 6.17: Inclusão de uma linha de código no procedimento associado ao evento OnClick de um item de menu.

Definindo Procedimentos Associando Eventos Para Itens de Menu Pop-up

A associação de um evento a um item de menu pop-up é feita de forma semelhante à associação de um evento a um item de menu.

Se você tiver um item de menu pop-up que deva realizar a mesma tarefa de um item de menu comum, basta estabelecer uma correspondência entre o procedimento associado ao evento OnClick desse último e o evento OnClick do item de menu pop-up. A isso denominamos compartilhamento de eventos (dois objetos compartilham um mesmo evento). Esse é o caso, por exemplo, do item Finalizar do menu pop-up Sistema.

Para criar o procedimento associado ao evento OnClick do item Finalizar do menu pop-up Sistema, faça o seguinte:

1. Na caixa de seleção de objetos do Object Inspector, selecione o objeto PopupSistemaFinalizar, como mostra a Figura 6.18.
2. Selecione a página Events do Object Inspector, se ela já não estiver selecionada.
3. Associe o evento OnClick desse objeto ao evento OnClick do objeto SistemaFinalizar, como mostra a Figura 6.19. Dessa forma, o mesmo procedimento estará associado aos eventos OnClick dos objetos SistemaFinalizar e PopupSistemaFinalizar.

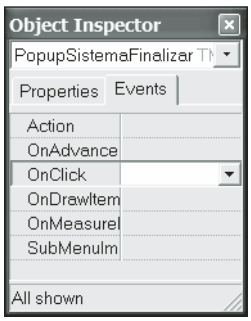


Figura 6.18: Selecionando o objeto PopupSistemaFinalizar no Object Inspector.

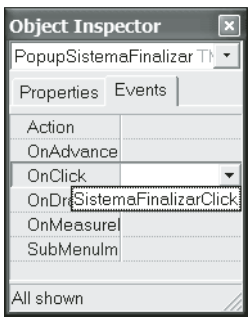


Figura 6.19: Definindo o procedimento associado ao evento OnClick do objeto PopupSistemaFinalizar.

Capítulo 7

Manipulando Formulários e Caixas de Diálogo

Neste capítulo serão apresentadas as técnicas de manipulação de caixas de diálogo, exemplificadas por meio da criação de uma caixa de diálogo usada para exibir informações sobre direitos autorais.

Fundamentos em: Criação de Caixas de Diálogo

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Conhecimentos básicos sobre a linguagem Object Pascal (já apresentados nos capítulos anteriores).

METODOLOGIA

- Apresentação e descrição das principais características de uma caixa de diálogo.

TÉCNICA

- Apresentação dos procedimentos necessários à exibição de caixas de diálogo a partir da janela principal de uma aplicação.

Caixas de Diálogo – Elementos de Interface Que Dão Vida ao seu Aplicativo

Geralmente uma aplicação com interface gráfica é composta por várias janelas e caixas de diálogos (que o Delphi 7 chama de formulários).

Neste capítulo serão apresentados os procedimentos necessários à criação de uma caixa de diálogo bastante simples e à sua exibição a partir da seleção de um item de menu.

Criando uma Caixa de Diálogo de Direitos Autorais

Para criar uma caixa de diálogo de direitos autorais, você pode usar um dos formulários predefinidos do Delphi 7, executando os seguintes procedimentos:

1. Selecione o item New/Other do Delphi 7. Será exibida a caixa de diálogo New Items, também denominada "Repositório de Objetos".

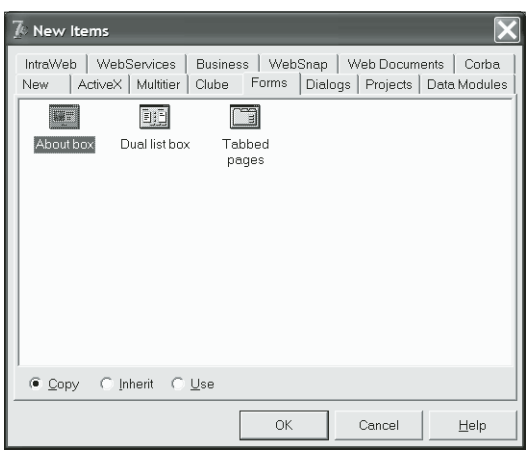


Figura 7.1: Seleção da opção About box da caixa de diálogo New Items.

2. Selecione a opção About box na página Forms da caixa de diálogo New Items, como mostra a Figura 7.1.
3. Clique no botão OK, para fechar essa caixa de diálogo e criar o novo formulário.

Será criada a caixa de diálogo About, mostrada na Figura 7.2.

Figura 7.2: A caixa de diálogo About.

Altere as propriedades Name e Caption desse formulário para FormSobre e Informações Gerais, respectivamente.

Repare que essa caixa de diálogo já possui os seguintes componentes:

- Um componente do tipo botão de comando, que o ambiente chamou de OkButton.
- Um componente de exibição de imagens, que o ambiente chamou de ProgramIcon.
- Quatro componentes do tipo Label para exibição de texto, que o ambiente chamou de ProductName, Version, Copyright e Comments.
- Um componente chamado Panel1, usado para criar um painel no formulário.

No próximo tópico mostraremos os procedimentos necessários à personalização desta caixa de diálogo.

Personalizando a Caixa de Diálogo de Direitos Autorais

Para personalizar a sua caixa de diálogo de direitos autorais, execute os seguintes procedimentos:

1. Altere o valor da propriedade Caption de ProductName para Cadastro de Sócios.
2. Altere o valor da propriedade Caption do label Version para Versão 1.0.
3. Altere o valor da propriedade Caption do label Copyright para Direitos Autorais.
4. Altere o valor da propriedade Caption do label Comments para o nome do autor do programa.
5. Altere o valor da propriedade WordWrap do label Comments para False.
6. Defina como True o valor da propriedade AutoSize de todos os labels (já deve estar definido).

7. Centralize horizontalmente todos os labels, usando a caixa de diálogo Alignment, apresentada no Capítulo 5.
8. Defina como True a propriedade AutoSize do componente ProgramIcon.
9. Carregue a imagem do arquivo Earth.bmp na propriedade Picture do componente ProgramIcon.
10. Salve o arquivo de código associado a este formulário com o nome UnitSobre.pas, usando o item Save As do menu File.

Seu formulário deve ficar como o da Figura 7.3.

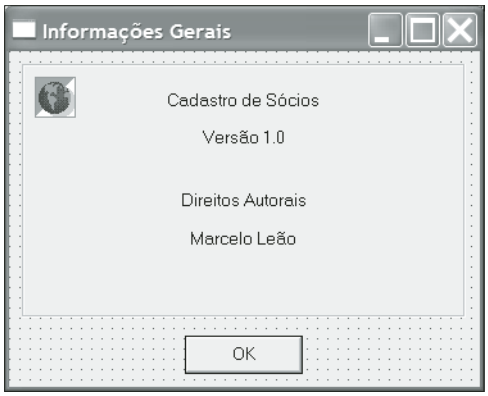


Figura 7.3: Aspecto final do formulário de direitos autorais.

Exibindo uma Caixa de Diálogo

No tópico anterior criamos uma caixa de diálogo chamada FormSobre, contendo informações de direitos autorais. No entanto, surge agora um pequeno problema: como exibir esse formulário? É evidente que esse formulário deverá ser exibido quando o usuário selecionar o item Sobre do menu Help.

Um formulário (ou uma caixa de diálogo) pode ser exibido de duas formas: como uma caixa de diálogo modal ou como uma caixa de diálogo não-modal. Uma caixa de diálogo modal é aquela que não permite que se execute qualquer ação no programa fora da caixa de diálogo até que esta seja fechada, ao passo que uma caixa de diálogo não-modal não impõe essa restrição. Um exemplo de caixa de diálogo modal é a caixa de diálogo Abrir do Microsoft Word, acessada por meio do item Abrir do menu Arquivo. Um exemplo de caixa de diálogo não-modal é a caixa de diálogo Localizar do Microsoft Word, acessada por meio do item Localizar do menu Edit.

Um objeto formulário apresenta dois métodos (herdados da classe TForm), denominados Show e ShowModal, que podem ser usados para exibi-lo. Para exibir uma caixa de diálogo modal, basta executar o método ShowModal do formulário, enquanto, para exibir uma caixa de diálogo não-modal, basta executar o seu método Show.

Inicialmente, temos de definir se essa caixa de diálogo será modal ou não-modal. Neste exemplo, vamos optar por uma caixa de diálogo modal, como geralmente acontece na maioria das aplicações.

No nosso caso queremos que, quando o item Sobre do menu Help for selecionado, essa caixa de diálogo seja exibida como uma caixa de diálogo modal.

Temos então a seguinte situação: quando o usuário selecionar o item Sobre do menu Help (quando ocorrer o *evento OnClick* desse item de menu), a caixa de diálogo deve ser exibida como uma caixa de diálogo modal (o *método ShowModal* do formulário deve ser executado).

Dessa maneira, devemos incluir a seguinte linha de código no procedimento associado ao evento OnClick do item Sobre do menu Help do formulário principal da aplicação:

```
FormSobre.ShowModal;
```

Para incluir essa linha de código no procedimento associado ao evento OnClick do item Sobre do menu Help, basta executar os seguintes procedimentos:

1. Exiba o formulário FormPrincipal e selecione o objeto HelpSobre (que é um item de menu), usando a caixa de seleção de objetos do Object Inspector.
2. Selecione a página Events do Object Inspector, se ela já não estiver selecionada.
3. Dê um duplo clique no campo à direita do evento a ser definido (no caso, o evento OnClick).
4. Inclua a linha de código acima, como mostra a Figura 7.4.

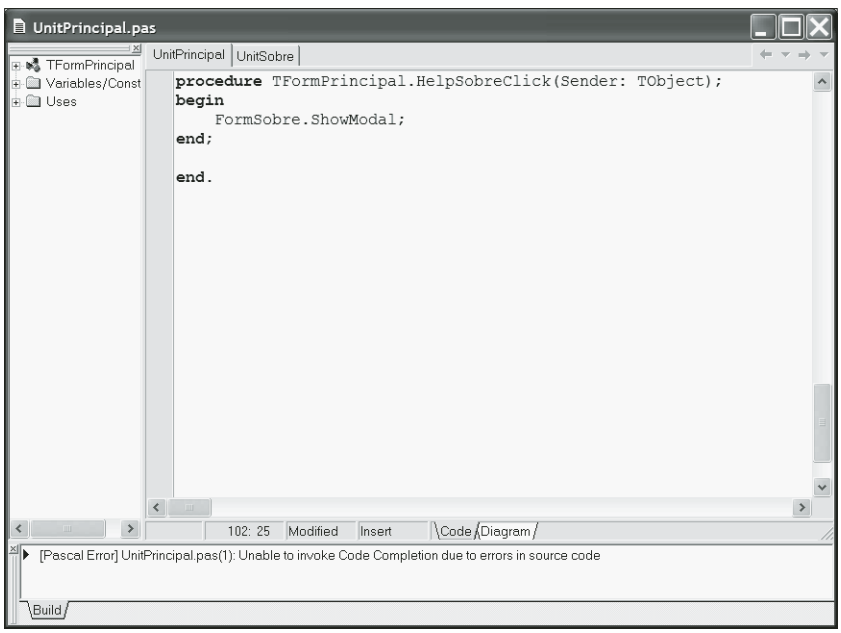


Figura 7.4: Inclusão de uma linha de código no procedimento associado ao evento OnClick de um item de menu.

5. Associe esse procedimento ao evento OnClick do item Sobre do menu pop-up criado no capítulo anterior.

Tente executar a sua aplicação. Infelizmente ela não compilará de imediato e será exibida uma mensagem de advertência (Figura 7.5), informando que o formulário FormPrincipal faz referência ao formulário FormSobre, declarado na unit UnitSobre, mas cujo nome não está incluído na cláusula uses da unit

que declarou o formulário Principal, e esta caixa de diálogo pergunta se você deseja incluí-la. Selecione o botão Yes, e o ambiente de desenvolvimento fará as alterações necessárias.

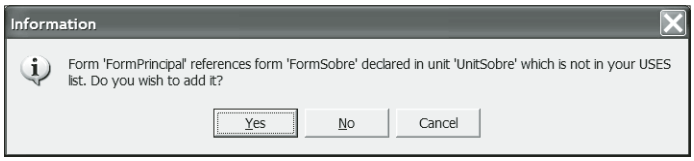


Figura 7.5: Mensagem de advertência do Delphi.

Tente executar outra vez a sua aplicação. Agora o programa funcionará normalmente e, quando o usuário selecionar o item Sobre do menu Help, a caixa de diálogo será exibida como uma caixa de diálogo modal, como mostra a Figura 7.6.

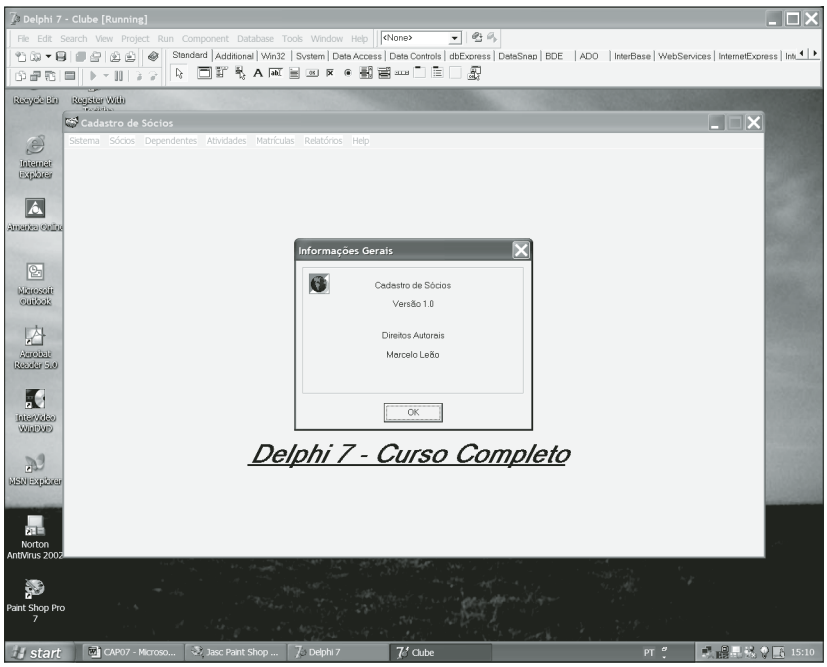


Figura 7.6: Exibindo uma caixa de diálogo modal.

No próximo tópico, vamos examinar o componente botão de comando, incluído automaticamente no formulário quando o mesmo foi criado, usando a caixa de diálogo New Items.

O Componente Botão de Comando

O botão de comando é um dos objetos vistos com maior freqüência nos aplicativos com interface gráfica. Ele apresenta formato retangular e normalmente se espera que algo aconteça quando é selecionado com o botão esquerdo do mouse.

O botão de comando (um objeto da classe TButton) está localizado na página Standard da paleta de componentes (é o sétimo componente desta página), e suas principais propriedades são apresentadas a seguir.

Principais Propriedades do Componente Botão de Comando

Entre as principais propriedades de um botão de comando, é possível destacar:

- Cancel: Essa propriedade, que pode assumir os valores True (verdadeiro) e False (False), indica se o botão deve associar o seu evento OnClick ao pressionamento da tecla Esc – isto é, se você definir o seu valor como True, pressionar a tecla Esc será equivalente a selecionar o componente com o botão esquerdo do mouse.
- Caption: Essa propriedade define o texto a ser exibido pelo botão.
- Default: Essa propriedade, que pode assumir os valores True (verdadeiro) e False (False), indica se o botão deve associar o seu evento OnClick ao pressionamento da tecla Enter – isto é, se você definir o seu valor como True, pressionar a tecla Enter será equivalente a selecionar o componente com o botão esquerdo do mouse.
- Enabled: Essa propriedade, que pode assumir os valores True (verdadeiro) e False (Falso), indica se o botão está habilitado. Quando o botão está desabilitado (a propriedade Enable tem o valor False), apresenta um aspecto acinzentado e não pode ser selecionado.
- Font: Define a fonte do texto exibido na propriedade Caption.
- ModalResult: Essa propriedade é usada para encerrar a execução de um quadro de diálogo modal. Você pode definir qualquer uma das constantes predefinidas do Delphi, como mrNone, mrOk, mrCancel, mrAbort, mrRetry, mrIgnore, mrYes, mrNo e mrAll.

Quando você criou o formulário FormSobre, o ambiente atribuiu o valor mrOk à propriedade ModalResult do componente OkButton. Dessa maneira, quando esse botão for selecionado com o botão esquerdo do mouse, o formulário será fechado.

Capítulo 8

Fundamentos do Projeto de Aplicativos de Banco de Dados

Neste capítulo, você será apresentado aos fundamentos do projeto de aplicativos de bancos de dados com o Delphi 7. Serão apresentados os fundamentos do mecanismo de acesso a bancos de dados multiplataforma denominado DBExpress, incorporado na versão 6 do Delphi.

Fundamentos em: Mecanismos de Acesso a Bancos de Dados

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Conhecimentos básicos sobre a linguagem Object Pascal (já apresentados nos capítulos anteriores).
- Utilização de utilitários para o ambiente Windows.

METODOLOGIA

- Apresentação e descrição dos principais conceitos relacionados ao desenvolvimento de aplicações que acessem bancos de dados, e seus mecanismos de acesso.

TÉCNICA

- Apresentação dos mecanismos de acesso a bancos de dados presentes no Delphi 7.

Mecanismos de Acesso a Bancos de Dados

Um dos fatores que impulsionaram o sucesso das versões anteriores do Delphi foi a capacidade de se criar uma aplicação capaz de acessar diferentes tipos de bancos de dados, com praticamente nenhuma alteração na sua interface. Uma mesma aplicação poderia ser desenvolvida para acessar tabelas Paradox e, com pequenas alterações, acessar tabelas do Interbase, do Access, SQL Server ou qualquer outro dos principais bancos de dados disponíveis no mercado. A escalabilidade de aplicações era uma realidade.

Isso se deve ao fato de, desde o lançamento da sua primeira versão, uma aplicação desenvolvida em Delphi não acessar diretamente um banco de dados, mas fazê-lo através de uma camada intermediária, denominada mecanismo de acesso.

Inicialmente, o único mecanismo de acesso disponível era o Borland Database Engine – BDE – um conjunto de DLLs desenvolvido pela própria Borland. A aplicação acessava o BDE e este acessava o banco de dados.

A partir da versão 5, o Borland Delphi passou a incluir componentes especiais, que permitiam que a aplicação acessasse outro mecanismo de acesso a banco de dados – o ADO (Activex Data Objects) da Microsoft. Além disso, foram também incluídos componentes para acesso nativo (e exclusivo) ao interbase (denominados Interbase Express).

O lançamento da versão 6 do Delphi (junto com o Kylix) trouxe um novo mecanismo de acesso – denominado DBExpress – presente nos dois ambientes de desenvolvimento – e que facilita o desenvolvimento de aplicações multiplataforma. O Delphi 6, no entanto, continuava a oferecer suporte ao BDE e ao ADO, e estes mecanismos continuam presentes na versão 7.

Embora este livro também aborde o acesso a bancos de dados via BDE e ADO, inicialmente será abordado o DBExpress, por ser multiplataforma e se tratar da nova aposta da Borland no que se refere a mecanismos

de acesso a bancos de dados (a Borland já anuncia, inclusive, que deixará de dar suporte ao BDE, embora este ainda esteja presente).

Fundamentos em: Conceitos Fundamentais Sobre Bancos de Dados

PRÉ-REQUISITOS

- Noções básicas sobre dados e informações.

METODOLOGIA

- Apresentação e descrição dos termos fundamentais relacionados à criação e acesso a bancos de dados.

Conceitos Fundamentais

Um banco de dados consiste em uma forma organizada de armazenamento de informações, mas seu conceito não representa uma inovação da era da informática. Há muito tempo as corporações (empresas, escolas, hospitais, órgãos governamentais, etc.) que necessitam manipular grande quantidade de informações armazenam dados de forma organizada e, antes que os custos da implantação de sistemas informatizados caíssem a níveis compatíveis com a realidade econômica das pequenas e médias empresas, estas organizavam seus dados e informações em enormes arquivos de aço.

A informatização trouxe, entre outros benefícios, a impressionante redução do espaço necessário ao armazenamento dessas informações e uma maior rapidez em sua consulta e pesquisa. Atualmente, muitas empresas disponibilizam um mesmo conjunto de informações em um banco de dados acessável localmente, pela Internet ou através de uma Intranet.

Antes da informatização, esses bancos de dados consistiam em enormes arquivos de aço utilizados para armazenar dados dos sócios, dados de atividades, etc.

Os dados dos sócios, por exemplo, eram guardados em fichas armazenadas em ordem alfabética, e cada ficha continha uma série de campos nos quais eram guardadas informações como nome, endereço, CPF, etc. Quando se cadastrava um novo sócio, uma nova ficha (um novo registro) deveria ser preenchida e armazenada (inserida) no local correto. Qualquer operação de busca de informações deveria ser feita dessa forma, e o armazenamento incorreto de uma ficha produzia enormes transtornos. O mesmo era válido para as informações das atividades, matrículas, etc.

Embora os antigos arquivos de aço tenham sido substituídos, muitos dos seus conceitos permanecem válidos. Os dados dos sócios continuam a ser armazenados em arquivos, porém estes são agora arquivos de dados armazenados em meios magnéticos especiais (disquetes, discos rígidos, CDs-ROM, Zip disks, DVDs, etc.). Esses arquivos também contêm fichas, que passaram a ser denominadas registros, e em cada registro (como nas fichas) existem diversos campos.

No caso do nosso arquivo de sócios, por exemplo, cada registro será usado para armazenar as informações de um determinado sócio, com um campo para o nome, outro para o endereço, etc.

Custos

Ao se desenvolver um aplicativo que acesse bancos de dados deve-se considerar também os custos envolvidos na aquisição de licenças que permitam a sua utilização de forma legal. Estes custos, no entanto, podem ser considerados elevados para pequenas empresas, e para estas situações há a alternativa de se empregar bancos de dados gratuitos, como a versão OpenSource do Interbase e o MySQL.

Neste livro abordaremos em nossos exemplos o Interbase, que por ser um produto da Borland e estar presente no próprio CD do Delphi, certamente estará ao alcance de nossos leitores.

Planejando seu Banco de Dados

Como já enfatizamos no Capítulo 4, um planejamento adequado é fundamental para o desenvolvimento de uma boa aplicação.

Neste livro, adotamos como exemplo o desenvolvimento de uma aplicação para gerenciamento de dados referentes aos sócios de um clube. Numa organização deste tipo, é comum que se armazenem informações a respeito dos sócios, atividades e matrículas.

Vejamos agora como estes dados devem ser organizados: cada sócio do clube pode se matricular em zero, uma ou mais atividades. Podemos então concluir que um sócio pode estar relacionado a várias atividades. A recíproca também é verdadeira, pois numa única atividade podem estar matriculados mais de um sócio.

Neste caso, diz-se que há uma relação de N x N entre sócios e atividades, e desta forma não seria muito razoável armazenar todos os dados de um sócio juntamente com os dados de cada uma das atividades (e vice-versa), pois estaríamos armazenando repetidamente (várias vezes) uma grande quantidade de informações.

A conclusão a que se chega é que os dados dos sócios e das atividades devem ser armazenados em tabelas distintas, e cada sócio deve ter um código exclusivo, que o diferencie dos demais. Na linguagem de banco de dados, um campo exclusivo em uma tabela (cujo valor não pode ser repetido) é denominado chave primária. Toda tabela deve ter ao menos um campo definido como chave primária, e quando isto não for possível pode-se usar mais de um campo, definindo-se o que se chama de uma chave primária composta. O mesmo se aplica às atividades.

Além disso, para cada sócio devem-se armazenar, além do seu código exclusivo, dados como o seu nome, seu CPF e seu endereço. Da mesma forma, para cada atividade devem-se armazenar também sua descrição e o valor da atividade.

Além das tabelas que armazenam os dados dos sócios e das atividades, devemos ainda criar uma tabela para armazenar os dados das matrículas. Nesta tabela devem ser definidos campos para armazenar o código do sócio e da atividade, (criando, desta forma, um relacionamento entre os sócios e as atividades.

Agora que já fizemos um esboço de como os dados serão armazenados, veremos no próximo capítulo como criar as tabelas que serão usadas no nosso aplicativo.

Nesta edição do livro optei por criar, nesta primeira parte, uma aplicação extremamente simples, de modo a facilitar a assimilação dos conceitos fundamentais, sem a necessidade de se digitar códigos extensos e sujeitos a erros.

Capítulo 9

Fundamentos de Bancos de Dados

Neste capítulo você aprenderá a criar as tabelas do Interbase que serão acessadas pela nossa aplicação. A opção pelo Interbase considerou o fato de que há uma versão que acompanha o próprio Delphi, e por ser multiplataforma, facilitando a migração, para o ambiente Linux, de aplicações desenvolvidas com base na biblioteca CLX e nesta base de dados.

Fundamentos em: Criação de Tabelas do Interbase

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Utilização de utilitários para o ambiente Windows.

METODOLOGIA

- Apresentação e descrição dos principais conceitos relacionados ao desenvolvimento de tabelas para bancos de dados do Interbase.

TÉCNICA

- Apresentação dos procedimentos necessários à criação de tabelas do tipo Interbase com o Database Desktop, utilitário para criação de tabelas que acompanha o Delphi 7.

Conceitos Fundamentais

Nesta primeira parte do livro são apresentados os procedimentos necessários ao desenvolvimento de aplicações que acessam localmente tabelas do Interbase.

Criando um Banco de Dados no Interbase

O Interbase é um banco de dados cliente-servidor. Nada nos impede, no entanto, de utilizá-lo como um banco de dados Desktop (ou local).

Para criar um banco de dados chamado Clube no Interbase, você deve executar os seguintes procedimentos:

1. Iniciar o aplicativo IBConsole, disponível no grupo de programas Interbase do Windows, e cuja tela principal é apresentada na Figura 9.1.
2. Selecione o item Local Server, no painel esquerdo do IBConsole, como mostrado na Figura 9.2.
3. Dê um duplo clique sobre o item selecionado ou selecione o item Login do menu Server, para exibir a caixa de diálogo Server Login mostrada na Figura 9.3.
4. Nesta caixa de diálogo, você deve, inicialmente, usar o nome de usuário (username) SYSDBA e a senha (password) masterkey, que já estão preconfigurados.
5. Selecione o botão Login para fechar esta caixa de diálogo e se conectar ao servidor local.

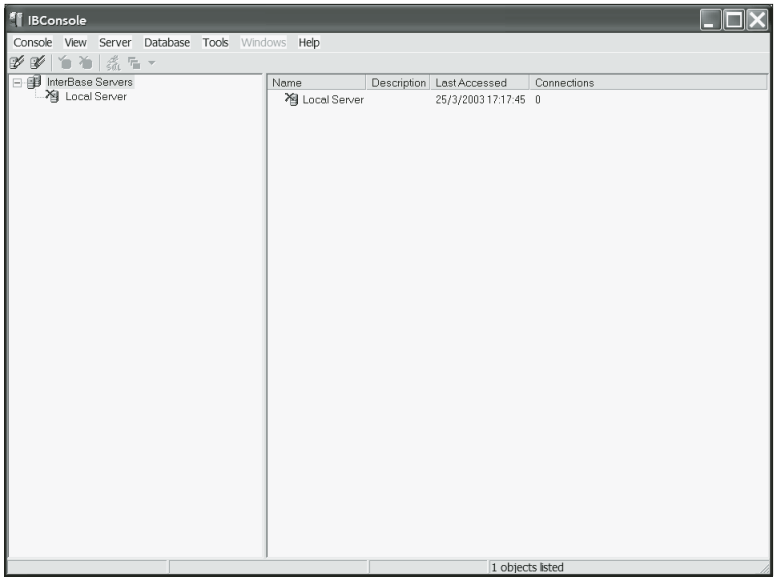


Figura 9.1: A tela principal do IBConsole.

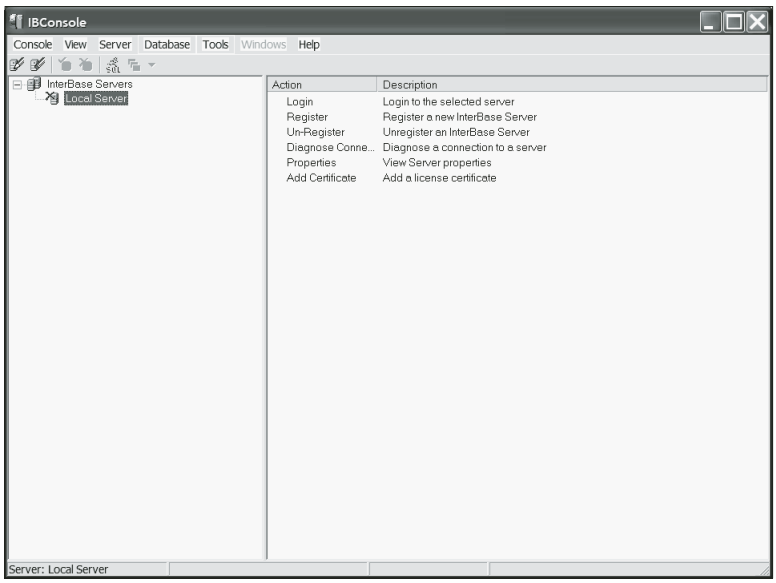


Figura 9.2: Selecione o item Local Server no painel esquerdo do IBConsole.

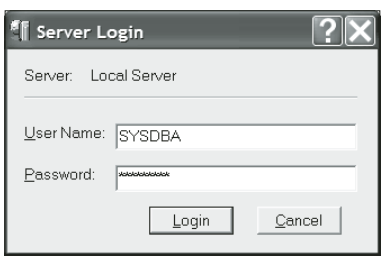


Figura 9.3: A caixa de diálogo Server Login.

Um servidor pode conter diversos bancos de dados. Repare que neste caso a conexão foi feita ao servidor, e não a um banco de dados.

6. Selecione o item Create Database, do menu Database, para exibir a caixa de diálogo mostrada na figura a seguir, na qual são fornecidas as informações necessárias à criação do banco de dados. Você deverá informar o nome do banco de dados (com extensão gdb, que é a extensão padrão para um banco de dados do Interbase) a ser criado e um alias (neste caso, CLUBE).

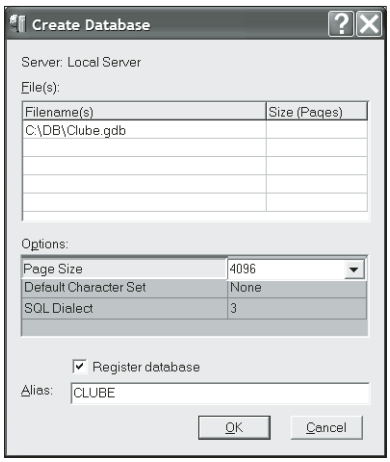


Figura 9.4: A caixa de diálogo Create Database.

7. Selecione o botão Ok, para fechar a caixa de diálogo e criar o banco de dados. O painel esquerdo do IBConsole mostra o banco de dados que acabou de ser criado (identificado pelo seu alias), como mostra a Figura 9.5.

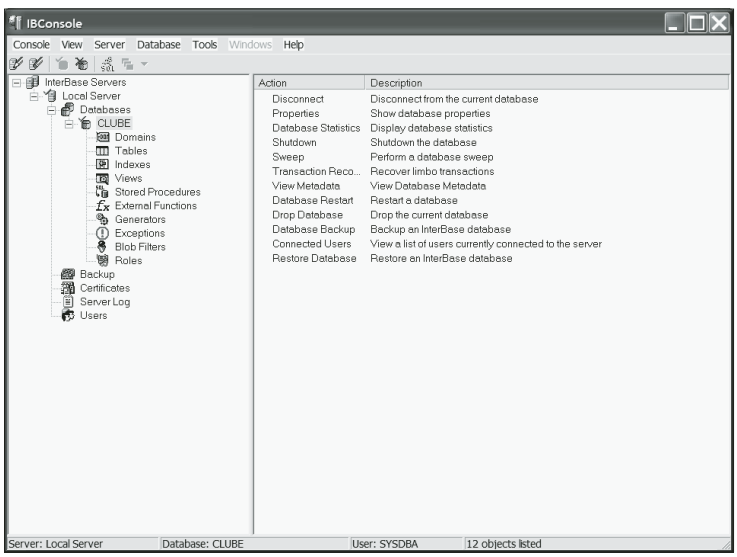


Figura 9.5: Criação do banco de dados CLUBE.

8. Com o banco de dados CLUBE selecionado, selecione o item Disconnect do menu Database. Será exibida uma mensagem solicitando uma confirmação para fechar o banco de dados, como mostra a Figura 9.6.

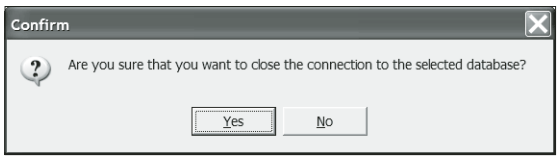


Figura 9.6: Mensagem de confirmação.

Criando Tabelas com o Database Desktop

O Database Desktop é um utilitário independente que acompanha o Borland Delphi 7, mas que pode ser acessado a partir de seu ambiente de desenvolvimento. Será utilizado por estar disponível e por simplificar o processo de criação de tabelas de bancos de dados do Interbase para o ambiente Windows.

Para iniciar o Database Desktop a partir do ambiente de desenvolvimento integrado do Borland Delphi 7, basta selecionar o item Database Desktop no menu Tools.

Será exibida a janela principal do Database Desktop, como mostra a Figura 9.7.

Figura 9.7: A janela principal do Database Desktop.

Antes de criarmos uma tabela para o nosso banco de dados, devemos criar um Alias (nome alternativo) para o mesmo, o que pode ser feito a partir do próprio Database Desktop. Este alias não deve ser confundido com aquele usado pelo IBConsole.

Para criar um alias para o banco de dados Clube com o Database Desktop, execute os seguintes procedimentos:

1. Selecione o item Alias Manager no menu Tools do Database Desktop. Será exibida a caixa de diálogo mostrada na Figura 9.8.

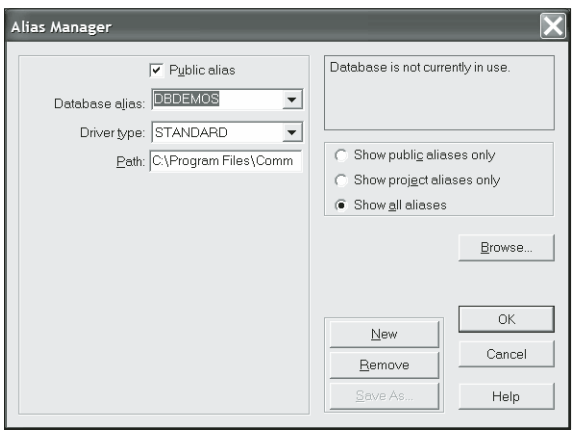


Figura 9.8: A caixa de diálogo Alias Manager do Database Desktop.

2. Selecione o botão New para criar um novo Alias, e configure-o como mostrado na Figura 9.9.
3. Selecione o botão Ok, para criar o alias. Será mostrada a mensagem de confirmação mostrada na Figura 9.10. Selecione Yes para confirmar a sua criação.

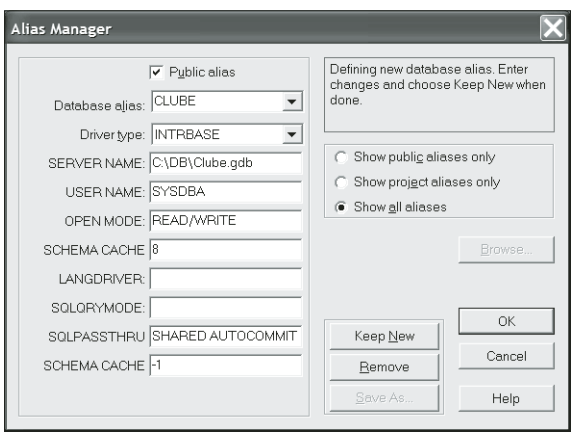


Figura 9.9: Configurando o Alias na caixa de diálogo Alias Manager do Database Desktop.

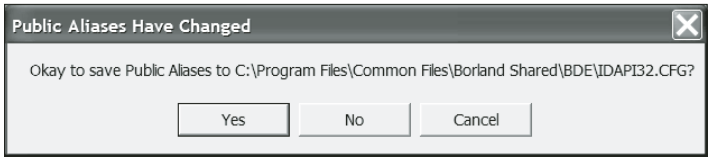


Figura 9.10: Confirmando a criação do Alias.

Para criar uma tabela com o Database Desktop, execute os seguintes procedimentos:

1. Selecione o item New do menu File. Será exibido um submenu, com as opções QBE Query, SQL File e Table, como mostra a Figura 9.11.

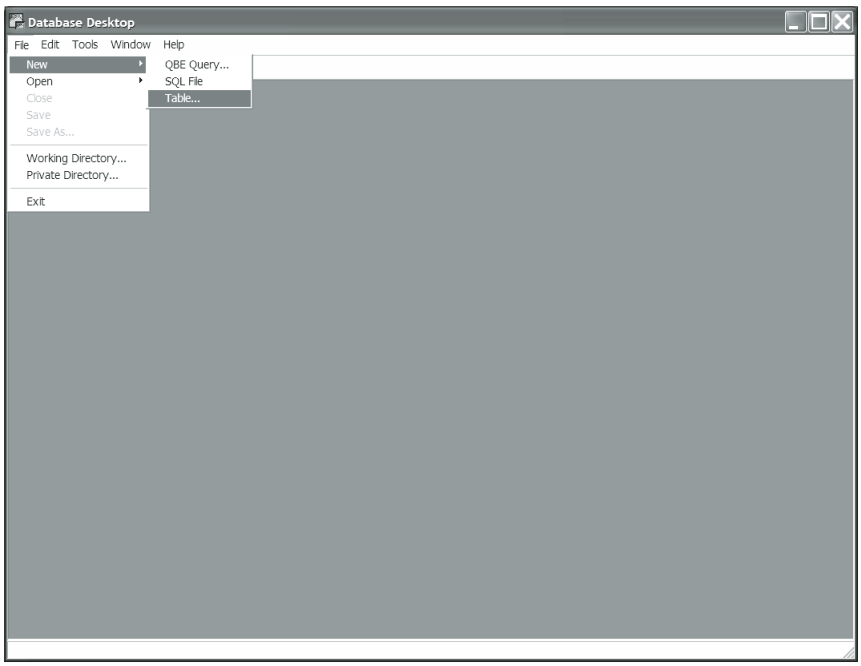


Figura 9.11: O submenu exibido quando o item New do menu File é selecionado.

2. Selecione o item Table do submenu File. Será exibida a caixa de diálogo Create Table, como mostra a Figura 9.12.

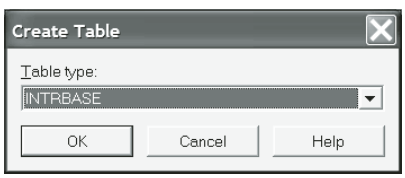


Figura 9.12: A caixa de diálogo Create Table.

A caixa de diálogo Create Table permite que você selecione os seguintes tipos de tabela:

- Paradox 7

- Paradox 5.0 for Windows
- Paradox 4
- Paradox 3.5
- Visual dBASE
- dBASE for Windows
- dBASE IV
- dBASE III+
- FOXPRO
- MS ACCESS
- SYBASE
- MSSQL
- INFORMIX
- INTRBASE
- DB2
- ORACLE

3. Selecione a opção INTRBASE, como mostrado na figura anterior, e depois clique no botão OK. Será exibida a caixa de diálogo Create INTRBASE Table, na qual podem ser definidos os campos que irão compor um registro da tabela (Figura 9.13).

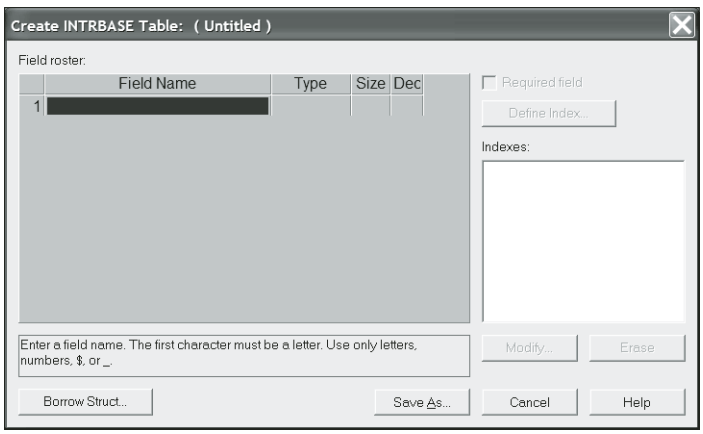


Figura 9.13: A caixa de diálogo Create INTRBASE Table.

Definindo Nomes Para os Campos dos Registros de uma Tabela

Para cada campo em um registro deve ser definido um nome (coluna Field Name), o tipo de dado que será armazenado no campo (coluna Type), seu tamanho (coluna Size) e o número de casas decimais (Dec).

Vamos começar a definir os nomes dos campos para a tabela que armazenará os dados dos sócios. Repare que, quando iniciamos a criação da nossa tabela, a coluna Field Name do primeiro campo aparece em destaque, indicando que o seu valor já pode ser digitado. Para definir o nome do primeiro campo, que armazenará o código do sócio, execute os seguintes procedimentos:

1. Digite a expressão CodigoSocio.
2. Pressione a tecla de seta para baixo.

Pronto! Repare que neste momento a coluna Field Name do segundo campo aparece em destaque, como mostra a Figura 9.14.

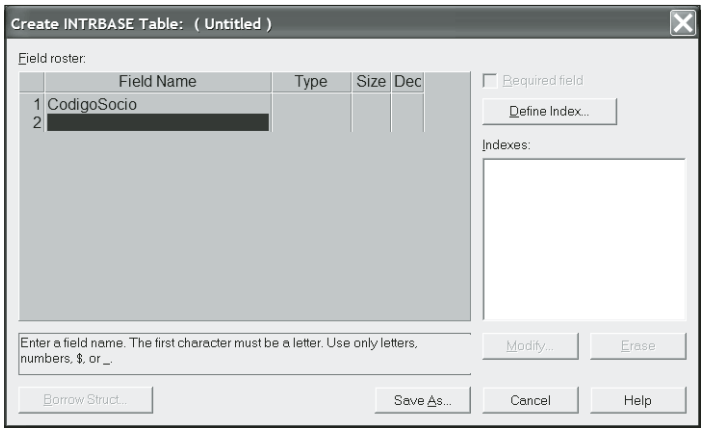


Figura 9.14: A caixa de diálogo Create INTRBASE Table após a definição de um nome para o primeiro campo.

Repita os passos anteriores para definir os seguintes nomes para os campos subseqüentes da tabela: Nome, Sobrenome, Endereço, Complemento, Bairro, Cidade, Estado, CEP, Telefone e CPF.

Após definir todos os nomes para os campos, a caixa de diálogo Create INTRBASE Table deverá ter a aparência mostrada na Figura 9.15.

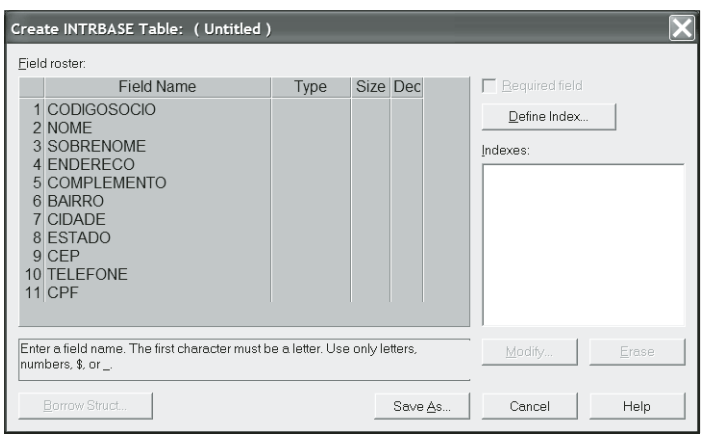


Figura 9.15: A caixa de diálogo Create INTRBASE Table após a definição de um nome para todos os campos.

Definindo Tipos Para os Campos dos Registros de uma Tabela

Conforme mencionado anteriormente, cada campo de um registro deve armazenar um valor de um determinado tipo. Para tabelas do tipo INTRBASE, os seguintes tipos podem ser utilizados:

- SHORT: Inteiros entre -32768 e 32767.
- LONG: Inteiros entre -2,147,483,648 e 2,147,483,648.
- FLOAT: Real de precisão simples (até 7 dígitos significativos de precisão).
- DOUBLE: Real de precisão dupla (até 15 dígitos significativos de precisão).
- VARCHAR(n): Armazena até *n* caracteres.
- DATE: Data
- BLOB: Objeto binário de tamanhoVariável.
- CHAR(n): Armazena *n* caracteres. (máximo 32767).

Os tipos disponíveis podem variar de um banco de dados para outro, e inclusive entre versões distintas de um mesmo banco de dados.

Para definir o tipo do campo CodigoCliente como LONG, execute os seguintes procedimentos:

1. Selecione com o botão esquerdo do mouse a coluna Type do campo CodigoSocio.
2. Digite LONG.

ou:

1. No menu pop-up exibido quando se pressiona o botão direito do mouse (Figura 9.16), selecione a opção LONG.
2. Pressione a tecla de seta para baixo.

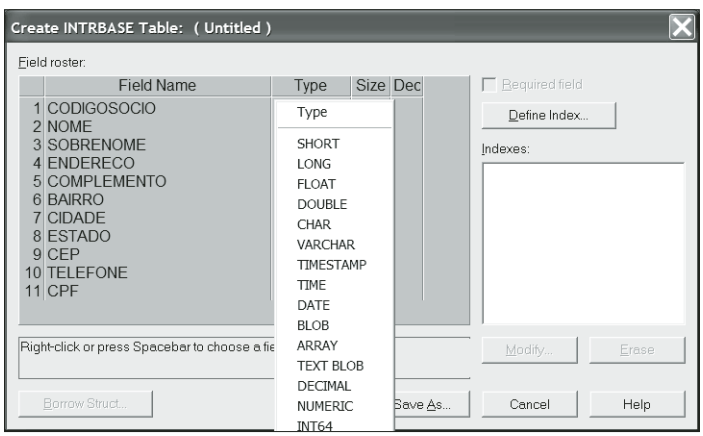


Figura 9.16: Menu pop-up, usado para selecionar um tipo para um campo.

Repita os passos anteriores para definir os demais campos como alfanuméricos (do tipo VARCHAR).

Alguns leitores podem pensar que o campo CEP, por exemplo, deveria ser definido como um campo numérico, pois armazena um grupo de números. Cabe salientar, no entanto, que, se o campo fosse definido como numérico, não poderia armazenar caracteres especiais como um hífen (-), que geralmente separa os cinco primeiros dígitos dos três últimos em um CEP, nem poderia ser criada uma máscara para exibição do campo (as máscaras de exibição serão vistas mais adiante). Além disso, como não será feita nenhuma operação aritmética sobre os valores armazenados neste campo, não há nenhum inconveniente em defini-lo como alfanumérico. Esta observação também se aplica aos campos CPF e Telefone.

Após definir os tipos de todos os campos, a caixa de diálogo Create INTRBASE Table deverá apresentar o aspecto mostrado na Figura 9.17.

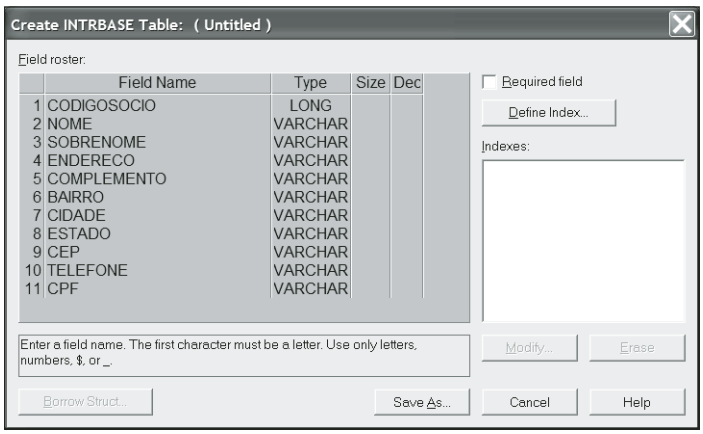


Figura 9.17: A caixa de diálogo Create INTRBASE Table após a definição de tipos para todos os campos.

Definindo os Tamanhos Para os Campos dos Registros de uma Tabela

Após definir o tipo de cada campo, deve-se definir o seu tamanho especificando-se o valor desejado na coluna Size.

Os campos alfanuméricos (VARCHAR) podem armazenar até 32767 caracteres. Logo, devemos armazenar na sua coluna Size o número máximo de caracteres que será permitido ao usuário digitar em um campo deste tipo.

Na nossa tabela de clientes, o primeiro campo é do tipo LONG, razão pela qual não será atribuído um valor para o seu tamanho na coluna Size.

Para definir o tamanho do campo Nome em 15 caracteres, execute os seguintes procedimentos:

1. Selecione com o botão esquerdo do mouse a coluna Size do campo Nome.
2. Digite o valor 15.
3. Pressione a tecla de seta para baixo, para passar para o próximo campo.

A caixa de diálogo Create Paradox INTRBASE deverá ficar com o aspecto indicado na Figura 9.18.

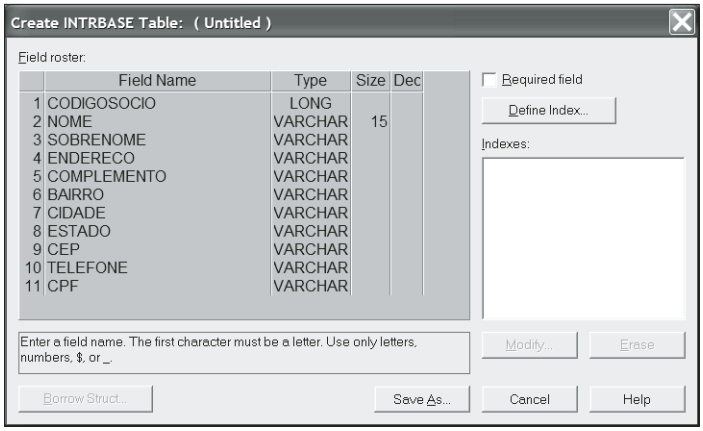


Figura 9.18: A caixa de diálogo Create INTRBASE Table após a definição do tamanho para o campo Nome.

Repita os passos anteriores, definindo os seguintes valores para os tamanhos dos demais campos:

- Sobrenome: 25.
- Endereço: 25.
- Complemento: 10.
- Bairro: 20.
- Cidade: 20.
- Estado: 2.
- CEP: 8.
- Telefone: 10.
- CPF: 11.

A caixa de diálogo Create INTRBASE Table deverá ficar com o aspecto indicado na Figura 9.19.

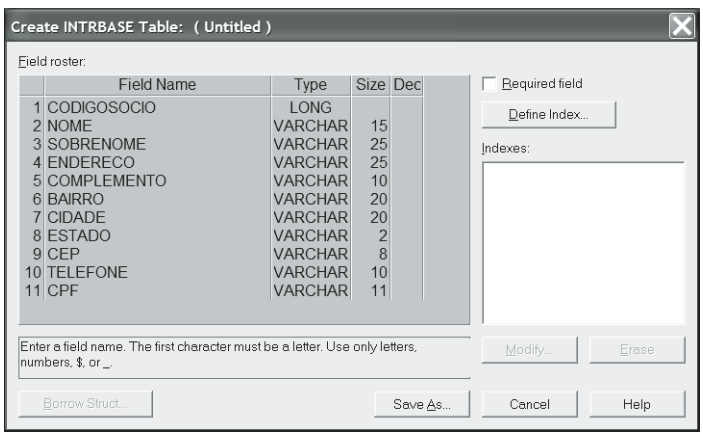


Figura 9.19: A caixa de diálogo Create INTRBASE Table após a definição de tamanho para os campos.

Definimos um tamanho igual a 10 para o campo telefone, considerando-se a nova modalidade de ligação interurbana. Um telefone do Rio de Janeiro poderia, por exemplo, ser cadastrado como (21)2345-6789, e não armazenaremos os caracteres não-numéricos.

Definindo Campos de Preenchimento Obrigatório

Para que os dados armazenados em um registro possuam significado, é necessário que seja fornecida uma quantidade mínima de informações, isto é, alguns campos devem ser obrigatoriamente preenchidos. Não faria muito sentido, por exemplo, cadastrar um sócio sem armazenar o seu nome e sobrenome, razão pela qual estes campos deverão ser preenchidos pelo usuário.

Para definir o campo Nome como um campo de preenchimento obrigatório, execute os seguintes procedimentos:

1. Selecione o campo Nome, clicando sobre o mesmo com o botão esquerdo do mouse.
2. Marque a caixa de verificação Required Field, como mostra a Figura 9.20.

Repita os passos anteriores para definir os campos CodigoSocio e sobrenome como campos de preenchimento obrigatório.

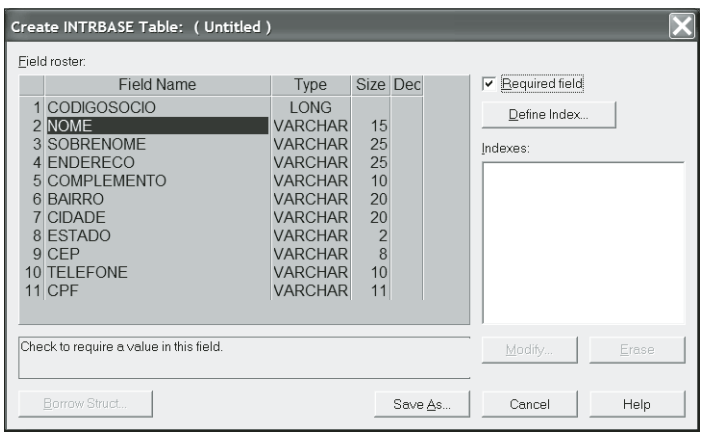


Figura 9.20: A caixa de diálogo Create INTRBASE Table após a definição do campo Nome como um campo de preenchimento obrigatório.

Criando Índices

Quando desejamos ordenar ou filtrar os registros de uma tabela por um determinado campo (ou grupo de campos), devemos associá-lo a um índice. Cada índice deve ter um nome que o identifique, como por exemplo ind_Nome para um índice correspondente ao Nome de um cliente. É recomendável que se atribuam aos índices nomes fáceis de memorizar, para auxiliar durante a fase de codificação do aplicativo.

Um índice pode ser simples, no caso de só corresponder a um campo, ou composto, no caso de se referir a mais de um campo.

No caso da tabela sócios, vamos definir os seguintes índices:

- Socios_Nome: Um índice composto, correspondente aos campos Nome e Sobrenome.
- Socios _Bairro: Um índice simples, correspondente ao campo Bairro.
- Socios _Cidade: Um índice simples, correspondente ao campo Cidade.

Para definir o índice Socios _Nome, execute os seguintes procedimentos:

1. Selecione o botão Define Indexes. da caixa de diálogo Create INTRBASE Table., para exibir a caixa de diálogo Define Secondary Index, mostrada na Figura 9.21.

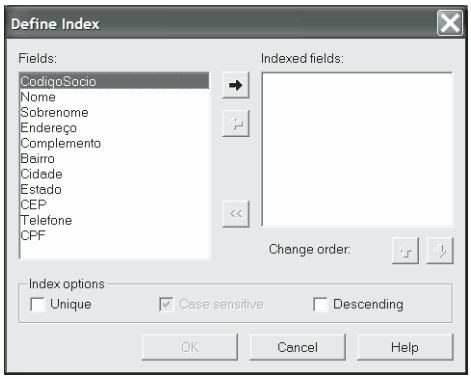


Figura 9.21: A caixa de diálogo Define Index.

2. Selecione com o mouse o campo Nome na caixa de lista Fields.
3. Pressione a tecla Shift e, mantendo-a pressionada, selecione o campo Sobrenome (para selecionar um campo sem desmarcar outros que já tenham sido previamente selecionados, mantenha pressionada a tecla Ctrl). A Figura 9.22 mostra os campos selecionados na caixa de lista Fields.
4. Selecione com o botão esquerdo do mouse o pequeno botão que exibe uma seta para a direita, para movimentar os campos selecionados para a caixa de lista Indexed Fields, como mostra a Figura 9.23.

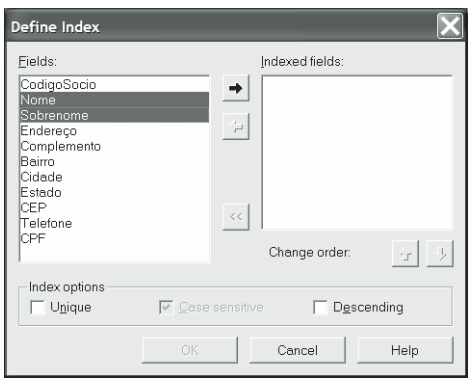


Figura 9.22: Selecionando os campos de um índice.

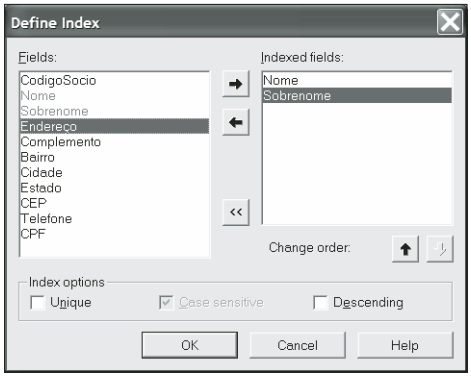


Figura 9.23: Movendo campos para a caixa de lista Indexed Fields.

5. Selecione o botão OK, e será exibida a caixa de diálogo Save Index As, na qual deve ser digitado o nome do índice, como mostra a Figura 9.24.

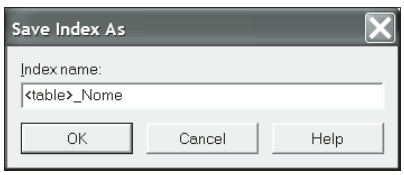


Figura 9.24: Definindo o Nome do índice.

Repare que a caixa de diálogo anterior exibe a expressão<Table>_, indicando que devemos apenas completar o nome do índice, pois o termo <Table> será substituído pelo nome da tabela. Neste caso, por exemplo, bastará informar <Table>_Nome.

Repita os passos anteriores para definir os índices simples Socios_Bairro e Socios_Cidade descritos anteriormente.

Repare que, após a sua definição, os nomes dos índices são exibidos na caixa de diálogo Create INTRBASE Table, como mostra a Figura 9.25.

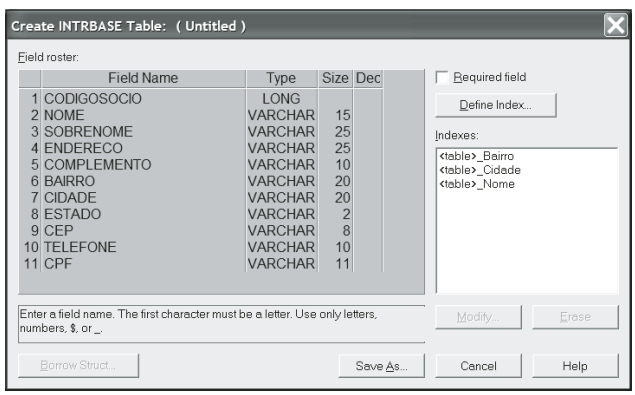


Figura 9.25: A caixa de diálogo Create INTRBASE Table após a definição dos índices.

Salvando uma Tabela

Para salvar a tabela de clientes, execute os seguintes procedimentos:

1. Selecione o botão Save As na caixa de diálogo Create INTRBASE Table para exibir a caixa de diálogo Save Table As, na qual devem ser definidos um nome para a tabela e o alias correspondente ao banco de dados no qual será armazenada, como mostra a Figura 9.26. No nosso exemplo, demos o nome Socios para a tabela e a armazenamos no banco de dados representado pelo alias Clube.
2. Clique no botão Salvar.

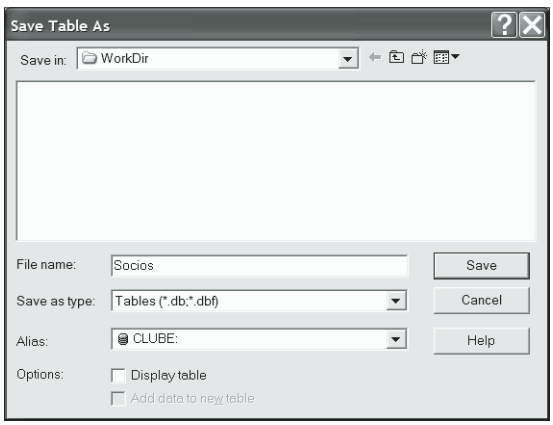


Figura 9.26: Salvando uma tabela com a caixa de diálogo Save Table As.

Se for exibida uma caixa de diálogo de Login, preencha com o username e password descritos anteriormente.

Pronto! Sua tabela foi salva.

Após salvar a tabela, o Interbase coloca todas as informações de nomes de campos e de índices em letras maiúsculas.

Inserindo Dados em uma Tabela Através do Database Desktop

Como já foi dito anteriormente, o Database Desktop é um utilitário independente, que permite a criação e o gerenciamento de tabelas de bancos de dados. Embora você esteja desenvolvendo um aplicativo que será capaz de fazer este tipo de gerenciamento de uma maneira mais personalizada, nada o impede de incluir uma massa de dados através do Database Desktop para testar alguns aspectos do seu aplicativo, antes mesmo de implementar toda a sua funcionalidade.

Caso queira inserir registros na sua tabela, basta colocá-la em modo de edição, o que pode ser feito executando-se os seguintes procedimentos:

1. Selecione o item Table do submenu Open do menu File do Database Desktop. Será exibida a caixa de diálogo Open Table, mostrada na Figura 9.27, na qual deverá ser selecionada a tabela a ser aberta clicando no botão Abrir (Não esqueça de especificar o Alias Clube).

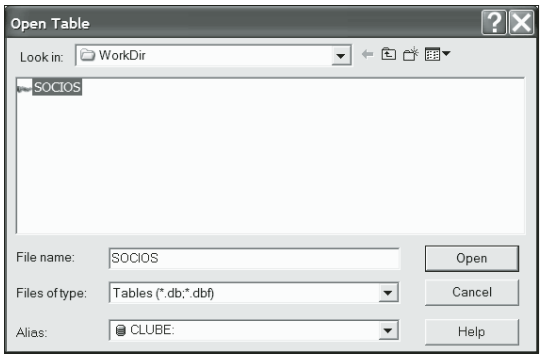


Figura 9.27: Abrindo uma tabela com a caixa de diálogo Open Table.

2. Selecione o item Edit Data do menu Table, ou pressione a tecla de função F9, para colocar a tabela em modo de edição, conforme indicado na Figura 9.28.

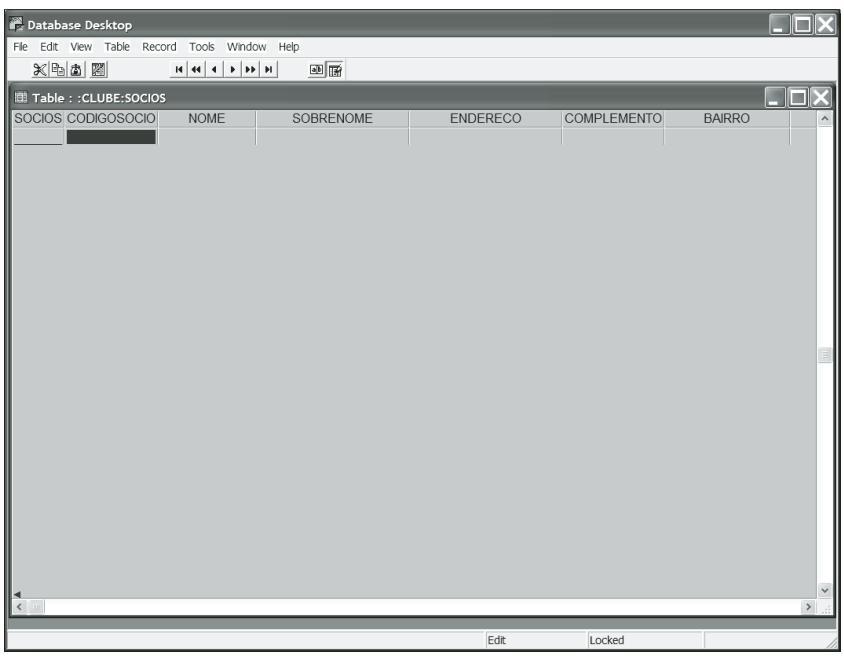


Figura 9.28: Colocando a tabela em modo de Edição.

Vamos incluir em nossa tabela os seguintes dados, referentes a três clientes (fictícios, naturalmente):

Cliente 1:

- CodigoCliente: 1
- Nome: Augusto.
- Sobrenome: Campos da Silva.

- Endereço: Rua dos Andradas 435.
- Complemento: Apto 905.
- Bairro: Cascadura.
- CEP: 22089-157.
- Cidade: Rio de Janeiro.
- Estado: RJ.
- Telefone: (21) 25768495
- CPF: 85725415608.

Cliente 2:

- CodigoCliente: 2
- Nome: Cláudia.
- Sobrenome: Moreira Bastos.
- Endereço: Avenida Santa Cecília 387.
- Complemento: Bl. B, 101.
- Bairro: Itaim.
- CEP: 05980555.
- Cidade: São Paulo.
- Estado: SP.
- Telefone: (11)37629645
- CPF: 15954235704.

Cliente 3:

- CodigoCliente: 3
- Nome: Paulo.
- Sobrenome: Aroeira dos Santos.
- Endereço: Avenida Guanabara 87.
- Complemento:
- Bairro: Grajaú.
- CEP: 22587344.
- Cidade: Rio de Janeiro.
- Estado: RJ.
- Telefone: (21)21225489
- CPF: 68575454601.

Para digitar os dados do primeiro cliente, faça o seguinte:

1. Digite o número 1 no campo CodigoCliente
2. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o campo Nome.
3. Digite a palavra "Augusto" no campo Nome.
4. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o campo Sobrenome.
5. Digite a expressão "Campos da Silva" no campo.
6. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o campo Endereco.
7. Digite a expressão "Rua dos Andradas 435" no campo.
8. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o campo Complemento.
9. Digite a expressão "Apto 905" no campo.
10. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o campo Bairro.
11. Digite a expressão "Cascadura" no campo.
12. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o campo Cidade.
13. Digite "Rio de Janeiro" no campo.
14. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o campo Estado.
15. Digite "RJ" no campo.
16. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o campo CEP.
17. Digite a expressão "22089157" no campo.
18. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o campo Telefone.
19. Digite a expressão "2125788495" no campo.
20. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o campo CPF.
21. Digite a expressão "85725415608" no campo.
22. Pressione a tecla Enter ou a tecla de seta para a direita para passar para o próximo registro.

Repita os procedimentos anteriores para inserir os dados dos outros dois clientes. Após inserir os dois novos registros, sua tabela deve ficar com o aspecto mostrado na Figura 9.29.

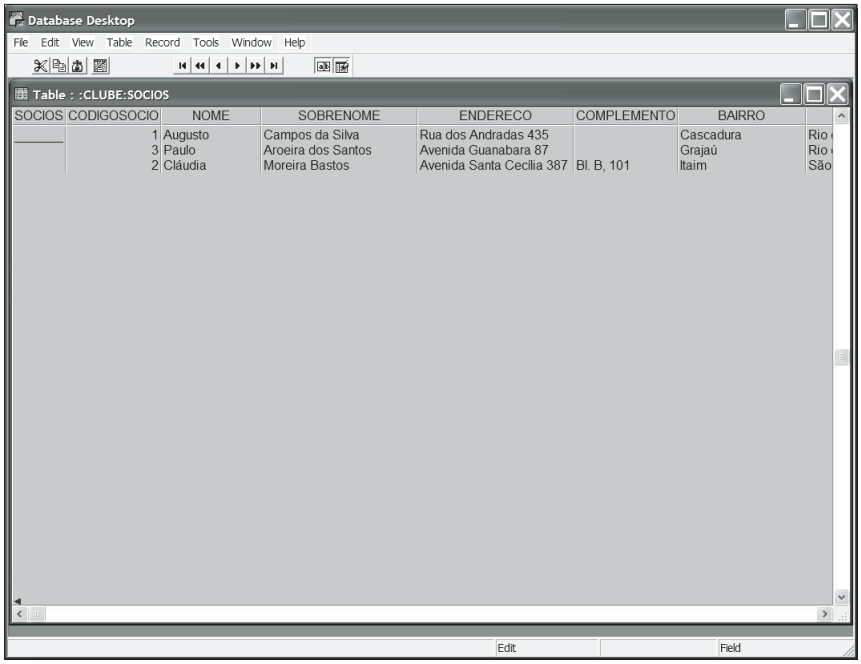


Figura 9.29: Aspecto da tabela após a inclusão dos três registros.

Construindo as Demais Tabelas do Aplicativo

Neste tópico serão apresentadas as características das demais tabelas que compõem o nosso aplicativo. Além da tabela Socios, serão necessárias as seguintes tabelas:

- Atividades, para armazenar os dados das atividades.
- Matriculas, para armazenar os dados das matrículas de sócios em atividades.

Criando a Tabela Atividades

Executando os procedimentos descritos nos tópicos anteriores, crie uma tabela chamada Atividades no banco de dados Clube, com os seguintes campos:

Nome do Campo	Tipo	Tamanho	Dec
CodigoAtividade	LONG		
Nome	VARCHAR	25	
Valor	Numeric	8	2

Após a definição dos campos indicados anteriormente, esta tabela deve apresentar, no Database Desktop, o aspecto mostrado na Figura 9.30.

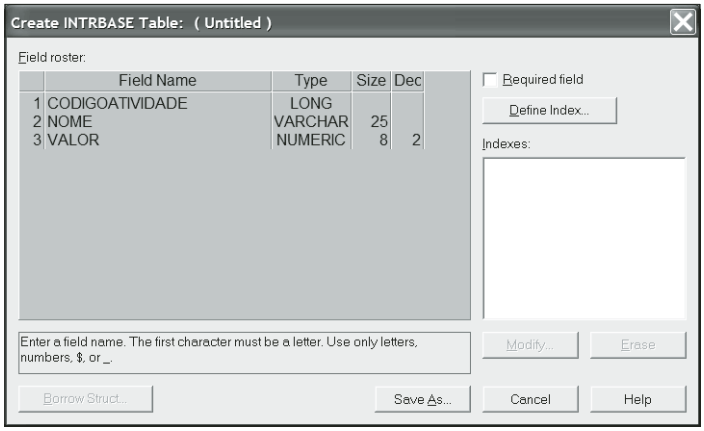


Figura 9.30: Criação da tabela Atividades.

Defina todos os campos desta tabela como campos requeridos.

Criando Índices Para a Tabela de Atividades

Para a tabela Atividades crie o seguinte índice, executando os procedimentos descritos nos tópicos anteriores:

- Atividade_Nome, para o campo Nome.

Salvando a Tabela Atividades

Salve a tabela no banco de dados representado pelo Alias CLUBE, com o nome Atividades, seguindo os mesmos passos descritos anteriormente para salvar a tabela Clientes.

Incluindo Registros na Tabela de Atividades

Coloque a tabela criada no tópico anterior em modo de edição e inclua os seguintes registros:

Atividade 1:

- CodigoAtividade: 1
- Nome: Natação.
- Valor: 30.00

Atividade 2:

- CodigoAtividade: 2
- Nome: Musculação.
- Valor: 25.00

Atividade 3:

- CodigoAtividade: 3

- Nome: Basquete.
- Valor: 40.00

Criando a Tabela de Matrículas

Executando os procedimentos descritos nos tópicos anteriores, crie uma tabela chamada Matriculas no banco de dados Clube, com os seguintes campos:

Nome do Campo	Tipo	Tamanho	Dec
Socio		LONG	
Atividade	LONG		

Defina estes dois campos como campos requeridos.

Nesta tabela o campo Socio, do tipo LONG, armazenará o código do sócio desta matrícula. Desta maneira, se precisarmos obter dados a respeito do sócio, basta consultar na tabela de sócios o registro cujo valor do campo CodigoSocio seja igual ao valor armazenado no campo Socio do registro corrente na tabela Matriculas. Veremos mais adiante como se estabelece um relacionamento entre tabelas, usando um campo como ligação entre elas. O mesmo se aplica ao campo Atividade.

Criando Índices Para a Tabela de Matrículas

Para a tabela de Matrículas crie os seguintes índices, executando os procedimentos descritos nos tópicos anteriores:

- Matriculas_Socio, para o campo Socio.
- Matriculas_Atividade, para o campo Atividade.

Pronto! As tabelas do nosso banco de dados já estão criados, e no próximo capítulo serão apresentados os componentes que permitem o acesso a estas tabelas a partir de aplicações desenvolvidas com o Delphi 7.

Capítulo 10

Criação de um Formulário Para Manipulação de Tabelas de Bancos de Dados com o DBExpress

Neste capítulo serão apresentados os procedimentos necessários à criação de um formulário para acesso a banco de dados no Delphi 7 usando como mecanismo de acesso o DBExpress. Serão utilizados os componentes das paletas Data Access e DBExpress (que, como já descrito anteriormente, é um mecanismo de acesso a dados multiplataforma).

Criação de Formulários Para Acesso a Dados

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Conhecimentos básicos sobre a linguagem Object Pascal (já apresentados nos capítulos anteriores).
- Noções básicas sobre bancos de dados.

METODOLOGIA

- Apresentação e descrição dos principais conceitos relacionados à criação de formulários para acesso a bancos de dados.

TÉCNICA

- Apresentação e descrição dos procedimentos necessários à criação de formulários para acesso a dados com o Delphi 7.

Para criar um formulário para acessar os dados da tabela de Sócios, você deve executar os seguintes procedimentos:

1 Abra o projeto Socio.dpr.

2 Selecione o item New/Form do menu File do Delphi 7. Será criado um formulário em branco.

3. Altere a propriedade Name deste formulário para FormCadastraSocios.

4. Altere a propriedade Caption deste formulário para Cadastro de Sócios.

5. Coloque um componente SQLConnection, situado na página DBExpress da paleta de componentes (é o primeiro componente desta página). Este é um componente não-visual (logo, sua posição no formulário é indiferente), e representa um banco de dados acessado pelo formulário.

6. Dê um duplo clique sobre este componente para exibir a caixa de diálogo Connection Editor, mostrada na Figura 10.1 (esta caixa de diálogo também pode ser exibida selecionando o item Edit Connection Properties no menu pop-up exibido quando você seleciona o componente com o botão direito do mouse).

7. Crie uma nova conexão chamada Clube, selecionando o botão com o símbolo '+' desta caixa de diálogo. Será exibida a caixa de diálogo New Connection, mostrada na Figura 10.2

8. Configure esta caixa de diálogo como mostrado na Figura 10.3 e selecione o botão Ok para fechá-la. Repare, como mostrado na caixa de diálogo apresentada na figura a seguir, que a nova conexão foi selecionada para ser configurada. Configure esta conexão como indicado, e selecione o botão Ok.

Repare na Figura 10.3 que, para este exemplo, foi criada uma conexão chamada Clube, tendo sido configuradas, na caixa de diálogo DBExpress Connection Editor, as propriedades DriverName, Database, UserName e Password.

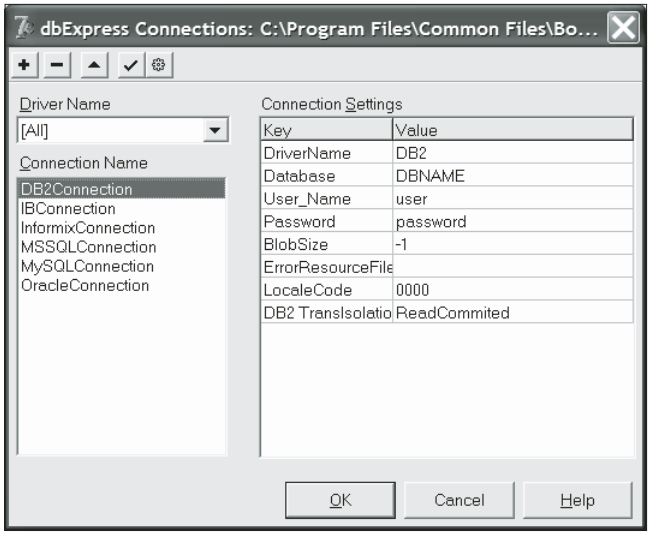


Figura 10.1: A caixa de diálogo DBExpress Connection Editor.

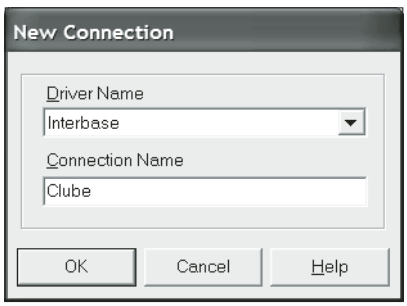


Figura 10.2: A caixa de diálogo New Connection.

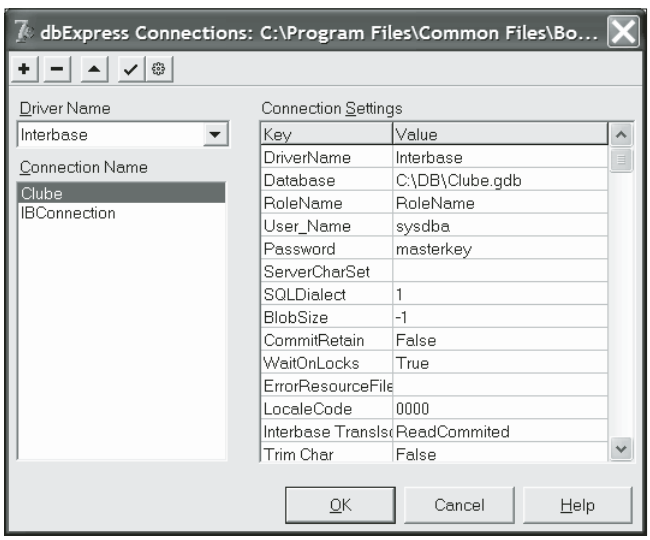


Figura 10.3: Criação da nova conexão.

Caso você queira usar uma conexão já definida para outro banco de dados do Interbase, basta alterar as propriedades da conexão, usando a caixa de diálogo Value List editor, exibida quando você seleciona os três pontinhos existentes à direita da propriedade Params do componente, mostrada na Figura 10.4.

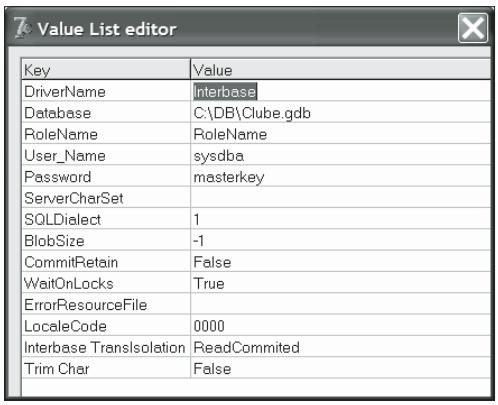


Figura 10.4: A caixa de diálogo Value List editor.

9. Atribua o valor True à propriedade Connected do componente. Se a propriedade LoginPrompt deste componente estiver configurada como True, será exibida a caixa de diálogo Database Login (na qual deverão ser informados o nome do usuário – UserName, e senha – password). Caso contrário, será feita uma tentativa de conexão com os valores default de User_Name e Password já fornecidos.
10. Após estabelecer a conexão ao banco de dados (definindo a propriedade Connected como True), altere o valor da propriedade Name do componente SQLConnection para SQLConnectionClube.
11. Insira no formulário um componente SQLTable, também situado na página DBExpress (é o quinto componente desta página).
12. Altere o valor da propriedade Connection deste componente para SQLConnectionClube.
13. Defina SOCIOS como o valor da propriedade TableName deste componente.
14. Altere o valor da propriedade Name deste componente para SQLTableSocios.
15. Altere para True o valor da sua propriedade Active.

O componente SQLTable é um componente não-visual, e que representa a tabela de Sócios no formulário.

16. Coloque um componente DataSource (disponível como o primeiro componente da página Data Access da paleta de componentes) no formulário e atribua o valor SQLTableSocios à sua propriedade Dataset. Este componente é responsável pela ligação entre os componentes de acesso a dados e os componentes de visualização (apresentados a seguir).
17. Altere o valor da propriedade Name deste componente para DataSouceSocios.

Seu formulário deverá ficar com o aspecto mostrado na figura a seguir.

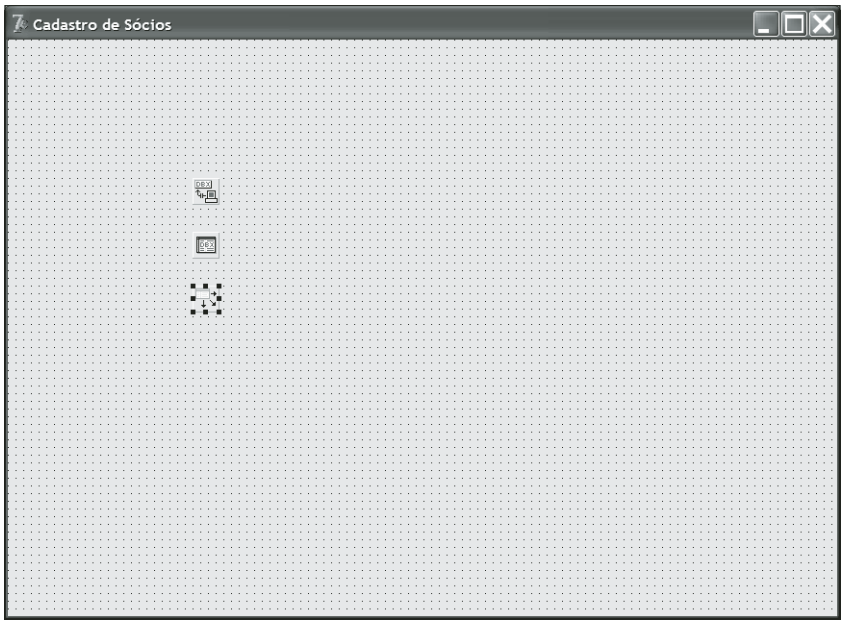


Figura 10.5: Aspecto do formulário, após a inclusão dos componentes de acesso a dados, com o componente Datasource selecionado.

Pronto! Já estão colocados os componentes de acesso a dados. Posteriormente abordaremos esta categoria de componentes em maiores detalhes.

Como os componentes inseridos até o momento são componentes não-visuais, sua posição no formulário é irrelevante.

18. Insira um componente Label (página Standard da paleta de componentes) no formulário, e altere o valor da sua propriedade Caption para Código.
19. Insira um componente DBEdit (quarto componente da página Data Controls da paleta de componentes) no formulário. Este componente, vinculado a uma tabela através de um componente Datasource, representa um dos campos do registro corrente. Suas principais propriedades são Name (que define o nome do componente), Datasource (que define o nome do componente Datasource ao qual está vinculado) e DataField (que define o nome do campo acessado pelo componente). Altere o valor da propriedade Name deste componente para DBEditCodigoSocio.
20. Altere o valor da propriedade Datasource deste componente para DatasourceClientes.
21. Altere o valor da propriedade DataField deste componente para CodigoSocio. Repare que, neste momento, o valor deste campo correspondente ao primeiro registro é exibido.
22. Repita os procedimentos anteriores de forma semelhante para cada um dos outros campos da tabela Clientes. Disponha os componentes de forma semelhante à mostrada na figura a seguir (ou com o layout que você julgar mais conveniente).

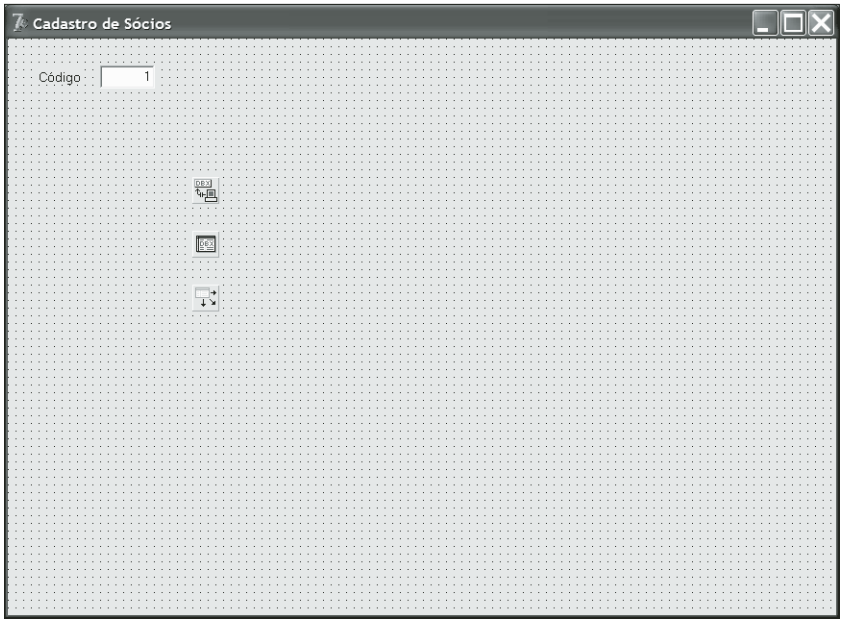


Figura 10.6: Aspecto do formulário, após a inclusão dos componentes de visualização.

23. Selecione o item Save As do menu File e salve-o com o nome UnitCadastroSocios.pas.
24. Inclua um botão de comando do tipo BitBtn (botão de figura, situado na página Additional da paleta de componentes) no formulário.
25. Altere os valores das propriedades Kind e ModalResult deste componente para bkCustom e mrNone, respectivamente.
26. Altere os valores das propriedades Name, Caption e Default deste botão de comando para BotaoCadastrar, &Cadastrar e False, respectivamente.
27. Inclua um segundo botão de comando do tipo BitBtn (botão de figura, situado na página Additional da paleta de componentes) no formulário.
28. Altere o valor da propriedade Kind deste componente para bkClose.
29. Altere os valores das propriedades Name e Caption deste botão de comando para BotaoFechar e &Fechar, respectivamente.

O fato de se definir o valor da propriedade Kind deste componente como bkClose fará com que, ao ser selecionado este botão, o formulário seja fechado. Além disso, se você alterar o valor da propriedade Kind depois de alterar o valor da propriedade Caption, o valor desta última propriedade será alterado para &Close.

30. Reposicione e redimensione os botões, de maneira que o mesmo fique com um aspecto semelhante ao apresentado na Figura 10.7.
31. Defina da seguinte maneira o procedimento associado ao evento OnShow deste formulário:

```
procedure TFormCadastraSocios.FormShow(Sender: TObject);
begin
    DBEditCodigoSocio.SetFocus;
    SQLTableSocios.Open;
```

```
    SQLTableSocios.Append;
end;
```

Neste procedimento, executado sempre que o formulário é exibido, são executados os métodos Open e Append do componente SQLTableSocios, da classe SQLTable. O método Open ativa o acesso à tabela. O método Append adiciona um novo registro em branco. Evidentemente, o método Open não é necessário se mantivermos a propriedade Active do componente como True. Desta forma, altere o valor da propriedade Active do componente SQLTableSocios para False. Faça o mesmo para a propriedade Connected do componente SQLConnectionClube.

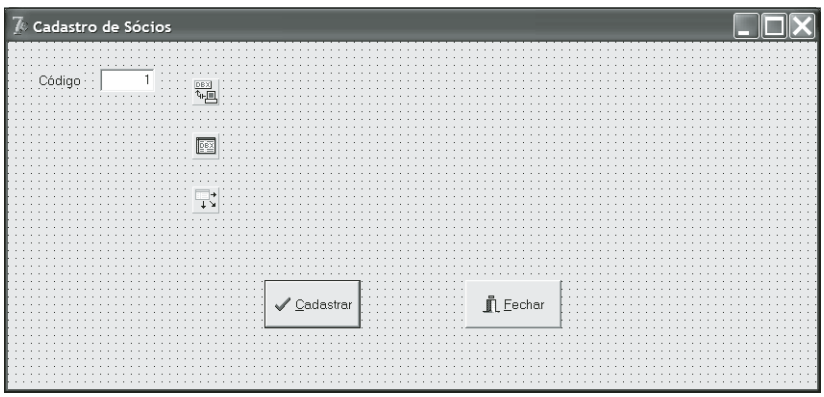


Figura 10.7: Aspecto do formulário, após a inclusão dos botões de comando.

Atribuir o valor True à propriedade Active do componente SQLTable equivale a executar o método open deste componente. Ao atribuir o valor True à propriedade Active do componente SQLTableSocios este tentará estabelecer a conexão através do componente SQLConnectionClube. Se a propriedade Connected deste componente estiver definida como False, o aplicativo tentará redefini-la como True.

32. Defina da seguinte maneira o procedimento associado ao evento OnClick do componente BotãoCadastrar:

```
procedure TFormCadastraSocios.BotaoCadastrarClick(Sender: TObject);
begin
    SQLTableSocios.Post;
    SQLTableSocios.Append;
    DBEditCodigoSocio.SetFocus;
end;
```

O método Post é responsável por gravar o último registro criado pelo método Append. Após a gravação do registro, um novo registro é criado com outra chamada ao método Append.

O método SetFocus do componente DBEdit é responsável por redirecionar o foco da aplicação (o cursor) para o componente DBEditCodigoSocio.

33. Defina da seguinte maneira o procedimento associado ao evento OnClose do formulário:

```
procedure TFormCadastraSocios.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    SQLTableSocios.Delete;
end;
```

Desta maneira, o último registro criado por um método Append, mas não gravado com um método Post, será apagado.

34. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item cadastro do menu Sócios.

```
FormCadastraSocios.Showmodal.
```

35. Compile a aplicação. Será exibida a mensagem de erro mostrada na figura a seguir, indicando que a unit unitCadastraSocios não foi incluída na cláusula uses da unit referente ao formulário principal. Selecione o botão Yes.

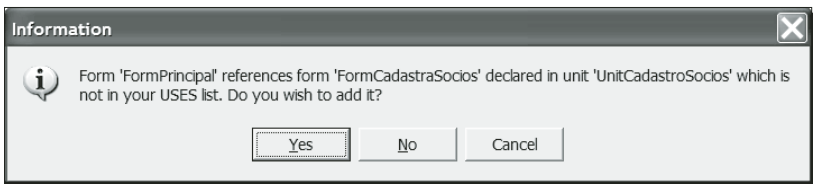


Figura 10.8: Mensagem de erro.

36. Execute e teste a aplicação. Ao selecionar o item Cadastro do menu Socios, será exibida a mensagem de erro mostrada na figura a seguir:

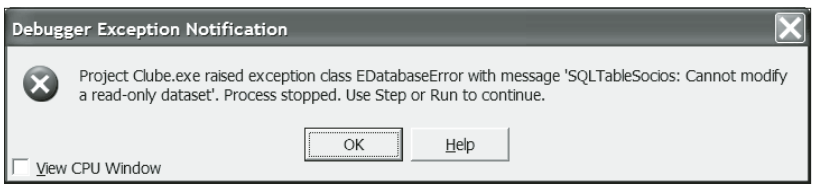


Figura 10.9: Mensagem de erro.

Esta mensagem indica que a aplicação está tentando editar uma tabela através de um componente disponível apenas para leitura. Esta é a primeira diferença do DBExpress para os demais mecanismos de acesso: O DBExpress (ao contrário do BDE e do ADO) é unidirecional, e não permite a edição direta dos registros (exceto em comandos SQL) – apenas a sua consulta, e desde que esta seja feita de forma unidirecional.

Para contornar este problema, devem-se utilizar os componentes DatasetProvider e ClientDataset (página Data Access da paleta de componentes), que em conjunto armazenam e gerenciam na memória o conjunto de registros acessados pelo componente SQLTable, e podem editar estes conjunto de registros (devendo posteriormente aplicar estas alterações ao banco de dados, mediante uma chamada ao método Applyupdates do componente ClientDataset).

No próximo tópico, serão apresentados os procedimentos necessários à utilização dos componentes DatasetProvider e ClientDataset.

Utilizando os Componentes DatasetProvider e ClientDataset

Para contornar o problema apresentado no tópico anterior, você deverá executar os seguintes procedimentos:

1. Inclua um componente DatasetProvider no formulário.
2. Altere as propriedades Name e Dataset deste componente para DatasetProviderSocios e SQLTableSocios, respectivamente.
3. Inclua um componente ClientDataset no formulário.
4. Altere as propriedades Name e ProviderName deste componente para ClientDatasetSocios e DatasetProviderSocios, respectivamente.
5. Altere a propriedade Dataset do componente DatasourceSocios para ClientDatasetSocios.
6. Defina como True o valor da propriedade Active do componente ClientDatasetSocios. Após realizar estas alterações, o formulário deverá ficar como mostrado na figura a seguir.

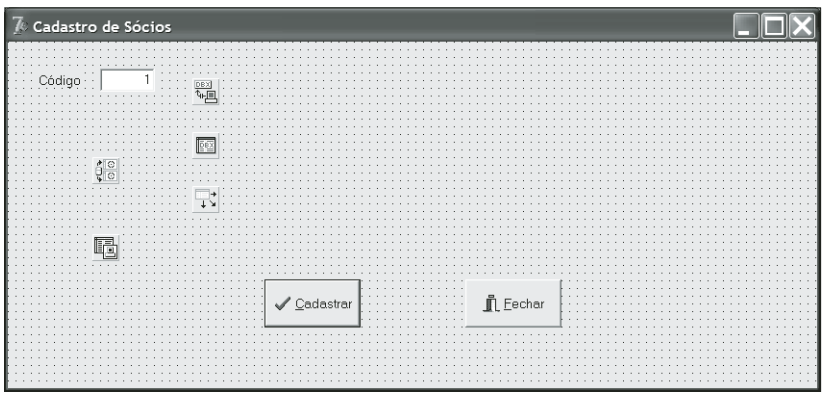


Figura 10.10: Aspecto do formulário, após a inclusão e configuração dos componentes DatasetProvider e ClientDataset.

7. Redefina da seguinte maneira o procedimento associado ao evento OnShow deste formulário:

```
procedure TFormCadastraSocios.FormShow(Sender: TObject);
begin
    DBEditCodigoSocio.SetFocus;
    ClientDatasetSocios.Open;
    ClientDatasetSocios.Append;
end;
```

8. Redefina da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoCadastrar:

```
procedure TFormCadastraSocio.BotaoCadastrarClick(Sender: TObject);
begin
    ClientDatasetSocios.Post;
    ClientDatasetSocios.Append;
    DBEditCodigoSocio.SetFocus;
end;
```

Desta forma, uma cópia dos registros será manipulada localmente pelo componente ClientDataset.

9. Redefina da seguinte maneira o procedimento associado ao evento OnClose do formulário:

```
procedure TFormCadastraSocios.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    ClientDatasetSocios.Delete;
    ClientDatasetSocios.ApplyUpdates(0);
end;
```

É importante destacar, neste caso, a chamada ao método ApplyUpdates do componente ClientDataset, responsável por efetivar no banco de dados as alterações realizadas no conjunto de registros manipulados localmente pelo componente. Lembre-se de que, ao ativar a conexão com o banco de dados, este componente recebeu uma cópia dos registros armazenados na tabela, e todas as inclusões, exclusões e alterações foram feitas nesta cópia. O método ApplyUpdates é responsável por gravar estas alterações no banco de dados, recebendo como parâmetro um número inteiro que identifica o número de erros permitidos em tal operação (entenda por erros as inconsistências entre os registros obtidos inicialmente e aqueles existentes no banco de dados no momento em que serão efetivamente registradas tais alterações (neste caso, definido como igual a zero).

10. Selecione o componente ClientDatasetSocios com o botão direito do mouse, para exibir o seu menu pop-up, como mostrado na figura a seguir.

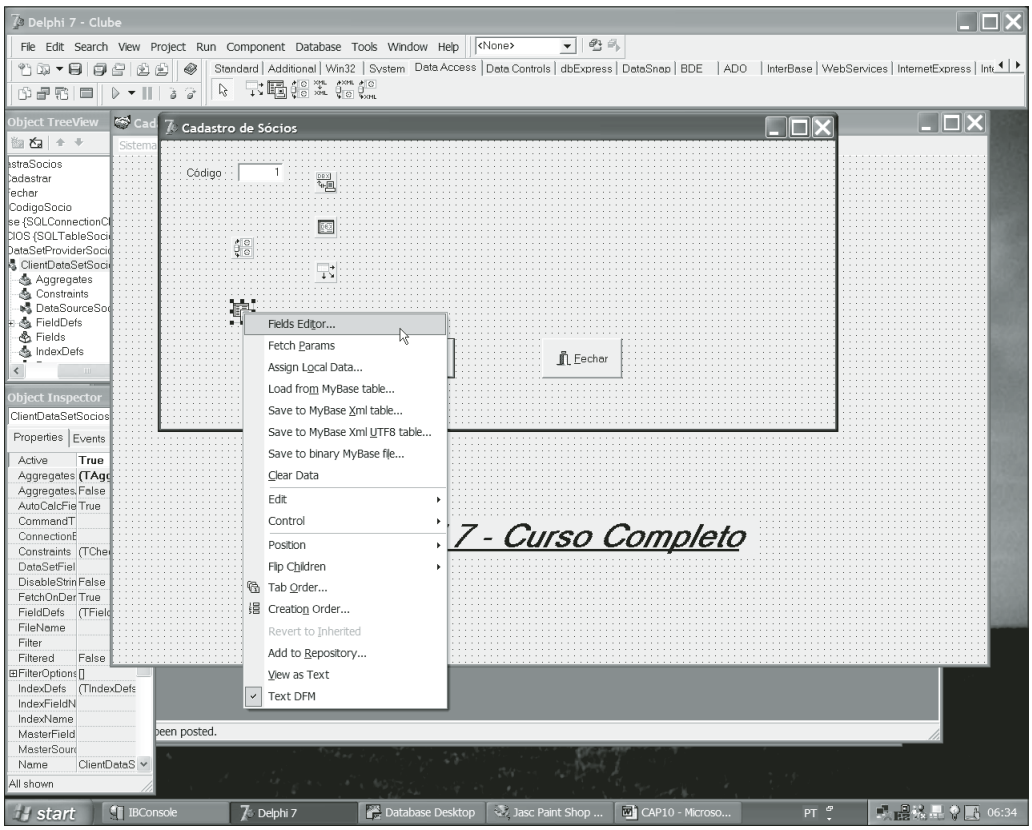


Figura 10.11: Acessando o menu pop-up do componente ClientDataset.

11. Selecione o item Fields Editor neste menu pop-up, para exibir a janela do editor de campos, mostrada na figura a seguir.

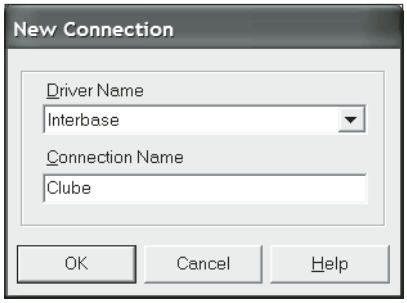


Figura 10.12: Acessando o editor de campos do componente ClientDataset.

12. Selecione com o botão direito do mouse a área branca desta janela para exibir o menu pop-up do Field Editor, mostrado na figura a seguir.

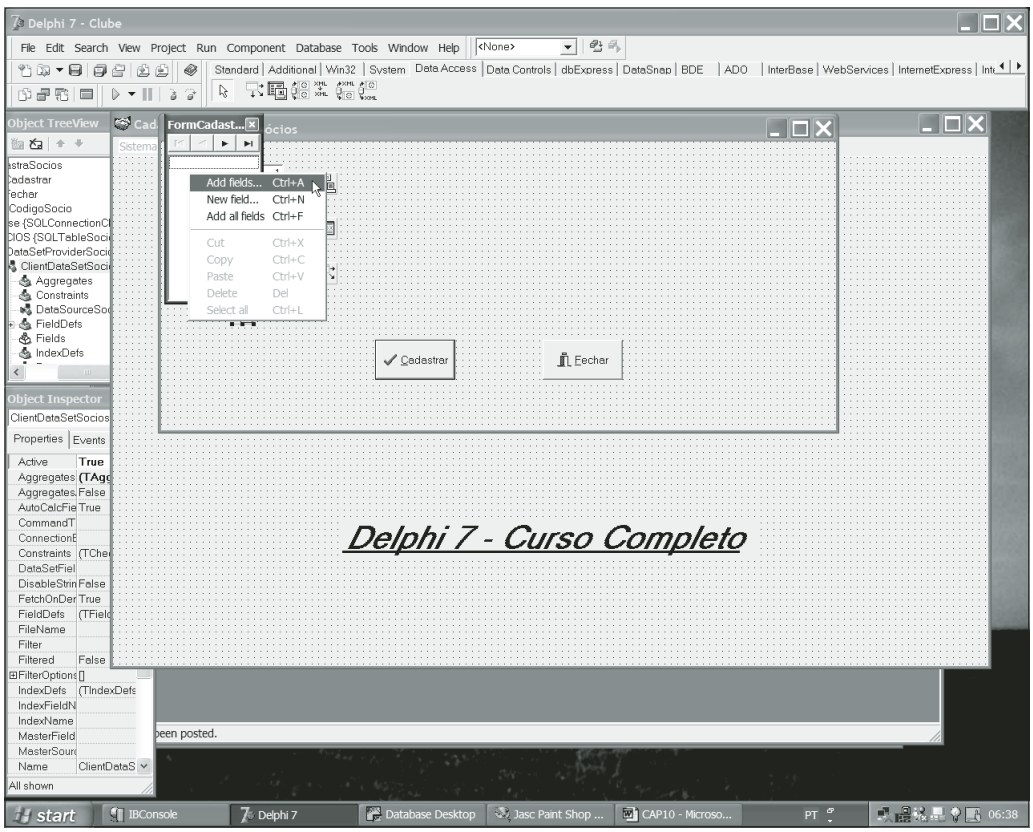


Figura 10.13: Acessando o menu pop-up do editor de campos do componente ClientDataset.

13. Neste menu popup, selecine o item Add All Fields. Serão criados os objetos que representam os campos, como mostrado na Figura 10.14.

14. A classe TFormCadastraSocios passa a incluir, entre seus campos, objetos que representam os campos da tabela, como mostrado no trecho de código reproduzido a seguir:

```
TFormCadastraSocios = class(TForm)
  SQLConnectionClube: TSQLConnection;
```

```
    SQLTableSocios: TSQLTable;
s    DataSourceSocios: TDataSource;
    Label1: TLabel;
    DBEditCodigoSocio: TDBEdit;
    BotaoCadastrar: TBitBtn;
    BotaoFechar: TBitBtn;
    DataSetProviderSocios: TDataSetProvider;
    ClientDataSetSocios: TClientDataSet;
    ClientDataSetSociosCODIGOSOCIO: TIntegerField;
    ClientDataSetSociosNOME: TStringField;
    ClientDataSetSociosSOBRENOME: TStringField;
    ClientDataSetSociosENDERECO: TStringField;
    ClientDataSetSociosCOMPLEMENTO: TStringField;
    ClientDataSetSociosBAIRRO: TStringField;
    ClientDataSetSociosCIDADE: TStringField;
    ClientDataSetSociosESTADO: TStringField;
    ClientDataSetSociosCEP: TStringField;
    ClientDataSetSociosTELEFONE: TStringField;
    ClientDataSetSociosCPF: TStringField;
    procedure FormShow(Sender: TObject);
    procedure BotaoCadastrarClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Repare que o campo CEP, por exemplo, é representado pelo objeto ClientDataSetSociosCEP, do tipo TstringField.

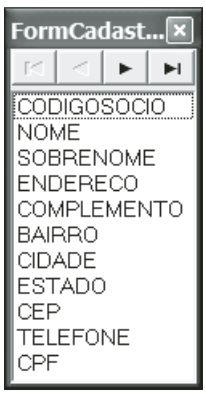


Figura 10.14: Incluindo os objetos que representam os campos da tabela no editor de campos do componente ClientDataset.

15. Inclua componentes Labels e DBEdits correspondentes aos demais campos da tabela de Sócios. Para isso, selecione no Fields Editor os objetos que representam os demais campos e "arraste" estes componentes para o formulário, como mostrado nas Figuras 10.15 e 10.16.

Reposicione e redimensione estes componentes para que seu formulário fique com o aspecto mostrado na Figura 11.17.

Atribua nomes adequados aos diversos componentes DBEdit, como DBEditSobrenomeSocio, DBEditEnderecoSocio, etc., para facilitar sua identificação no código da aplicação.

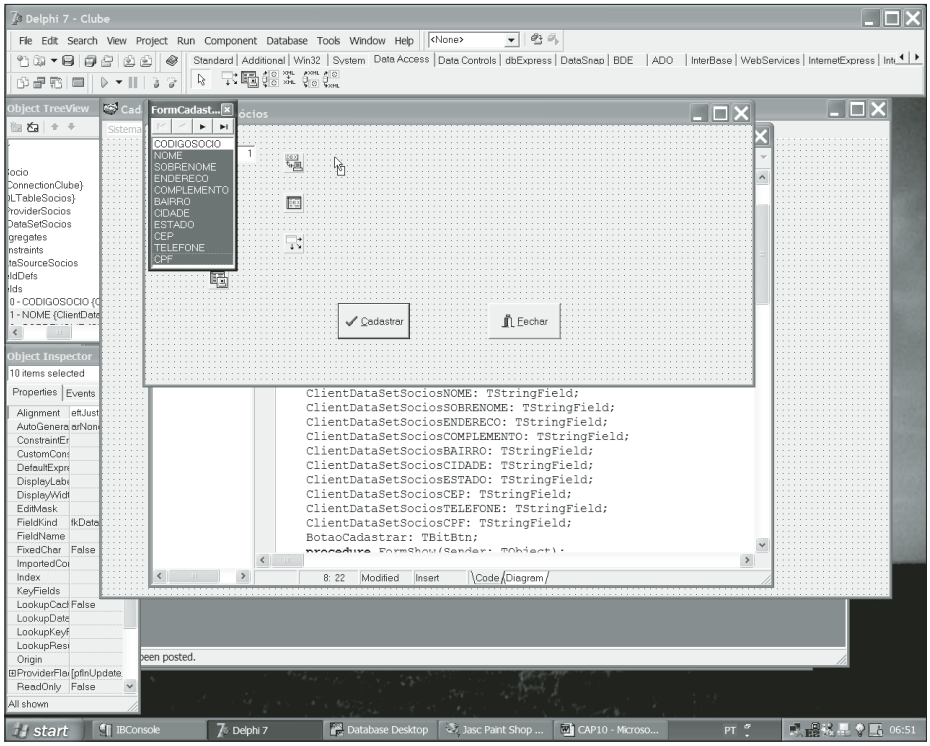


Figura 10.15: Arrastando os objetos que representam os demais campos da tabela de Sócios a partir do editor de campos para o formulário.

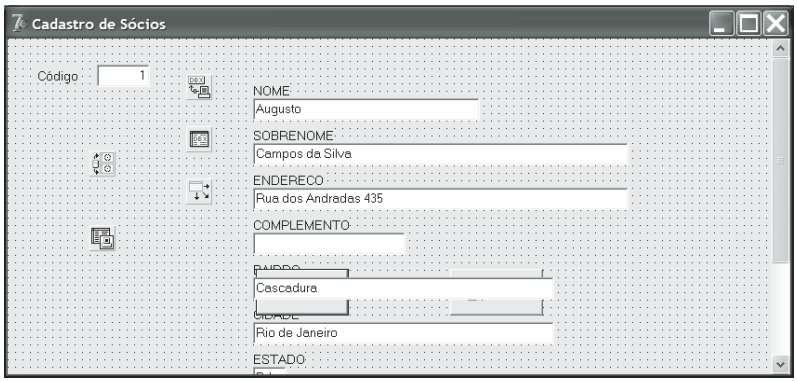


Figura 10.16: Criando os objetos para acessar os demais campos da tabela de Sócios a partir do formulário.

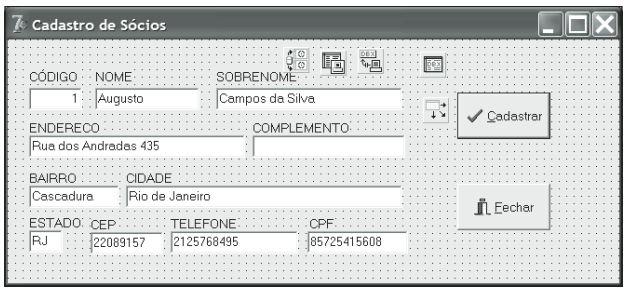


Figura 10.17: Criando os componentes para acessar os demais campos da tabela de Sócios a partir do formulário.

O Componente DBEdit

Este componente, usado anteriormente, exibe o valor de um campo do registro corrente e permite que o usuário altere o seu valor. Entre as propriedades deste componente, podemos destacar:

- DataSource: Esta propriedade define o componente que nos fornecerá a conexão a um componente que representa uma tabela ou um banco de dados.
- Name: Esta propriedade define o nome pelo qual o componente será referenciado no programa.
- DataField: Esta propriedade especifica o campo da tabela cujo valor será exibido.

No nosso exemplo, temos diversos controles deste tipo, todos com a propriedade DataSource igual a DataSourceSocios e cuja propriedade DataField vale, respectivamente: Nome, Sobrenome, Endereço, Complemento, Bairro, Cidade, Estado, CEP, Telefone e CPF. Neste caso, como usamos o Fields Editor, estas propriedades foram configuradas automaticamente quando os componentes foram criados no formulário.

O componente que exibe o valor do campo Nome, por exemplo, possui as seguintes propriedades:

- DataSource: DataSourceSocios
- DataField: Nome
- Name: DBEditNomeSocio

Definindo Máscaras Para os Campos

Uma deficiência que este formulário apresenta é a seguinte: nada impede que o usuário digite um valor como "abcde-fgh" ou "xasdr-auy", no campo CEP, pois ambos são strings de caracteres e podem ser armazenadas em campos alfanuméricos.

Para restringir os caracteres que o usuário pode digitar em um componente que representa um campo de uma tabela, devemos definir uma máscara para este campo.

Para definir uma máscara para um campo, você deve atribuir um valor à propriedade EditMask do objeto que representa o campo (criado com o editor de campos). Estes objetos estão definidos na classe do formulário, mas não serão visualizados como componentes, embora você possa acessá-los através do Object Inspector para definir suas propriedades (como a propriedade EditMask, por exemplo).

1. Usando a caixa de seleção de objetos do Object Inspector, selecione o objeto ClientDataSetSociosCEP, como mostra a Figura 10.18.

Uma vez selecionado o objeto que representa o campo, o próximo passo consiste em definir um valor para a sua propriedade EditMask.

A propriedade EditMask é constituída de três partes, separadas por ponto-e-vírgula. A primeira parte representa a máscara propriamente dita; a segunda parte define se apenas os caracteres digitados (valor igual a 0 – o default) ou toda a máscara deve ser armazenada na tabela (valor igual a 1); a terceira parte especifica o caractere usado para representar espaços em branco (o default é "_").

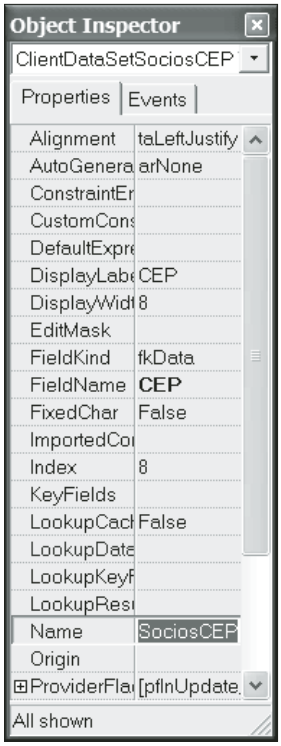


Figura 10.18: Selecionando o objeto ClientDataSetSociosCEP.

Significado dos Caracteres Usados na Propriedade EditMask

A seguir são mostrados os caracteres que podem ser usados na definição da propriedade EditMask:

- !: Se um! aparecer na máscara, caracteres em branco não são armazenados como dados.
- >: Se um > aparecer na máscara, todos os caracteres seguintes estarão em letras maiúsculas, até que seja encontrado um caractere igual a <.
- <: Se um < aparecer na máscara, todos os caracteres seguintes estarão em letras minúsculas, até que seja encontrado um caractere igual a >.
- <>: Significa que as letras maiúsculas e minúsculas serão armazenadas como digitadas pelo usuário.
- \: Os caracteres que seguem este sinal serão tratados como caracteres literais.
- L: Este caractere faz com que apenas caracteres alfabéticos (a-z e A-Z) sejam aceitos nesta posição.
- l: Este caractere faz com que apenas caracteres alfabéticos (a-z e A-Z) sejam aceitos nesta posição, mas aceita que sejam omitidos.
- A: Este caractere faz com que apenas caracteres alfanuméricos (0-9, a-z e A-Z) sejam aceitos nesta posição.
- a: Este caractere faz com que apenas caracteres alfanuméricos (0-9, a-z e A-Z) sejam aceitos nesta posição, mas aceita que sejam omitidos.
- C: Este caractere faz com que quaisquer caracteres sejam aceitos nesta posição.

- c: Este caractere faz com que quaisquer caracteres sejam aceitos nesta posição, mas aceita que sejam omitidos.
- 0: Este caractere faz com que apenas caracteres numéricos (0-9) sejam aceitos nesta posição.
- 9: Este caractere faz com que apenas caracteres numéricos (0-9) sejam aceitos nesta posição, mas aceita que sejam omitidos.
- #: Este caractere faz com que apenas caracteres numéricos (0-9) e sinais de mais (+) ou menos (-) sejam aceitos nesta posição, mas aceita que sejam omitidos.
- :: Este caractere (dois-pontos) é usado para separar horas, minutos e segundos em dados horários. Se outro caractere separador de horas, minutos e segundos for definido nos atributos internacionais do Painel de Controle do seu sistema, este será usado em lugar de dois-pontos.
- /: Este caractere é usado para separar meses, dias e anos que representam dados de datas. Se outro caractere separador de meses, dias e anos for definido nos atributos internacionais do Painel de Controle do seu sistema, este caractere será usado em lugar do caractere /.
- ;: O ponto-e-vírgula é usado para separar os pontos de uma máscara.
- _: O caractere _ insere espaços em branco na caixa de edição.

Você também pode definir o valor da propriedade EditMask através do Editor de máscaras. Para acessar este editor, selecione com o mouse as reticências (...) exibidas do lado direito da propriedade no Object Inspector. Será exibida a caixa de diálogo Input Mask Editor, como mostra a Figura 10.19.

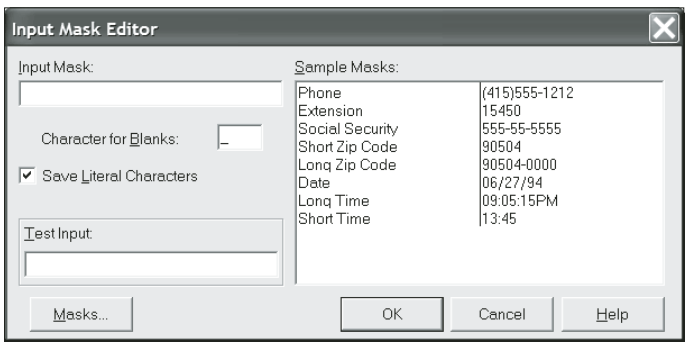


Figura 10.19: A caixa de diálogo Input Mask Editor.

Esta caixa de diálogo apresenta as seguintes características:

- Permite que você selecione uma das máscaras já disponíveis na lista Sample Masks.
- Permite que você personalize a máscara na caixa de texto Input Mask.
- Permite que você especifique o caractere usado para representar espaços em branco na caixa de texto "Character for Blanks".
- Permite que você defina se toda a máscara deve ser armazenada na tabela ou apenas os dados digitados pelo usuário, através da caixa de verificação Save Literal Characters.
- Permite que você teste a máscara, digitando valores na caixa de texto Test Input.

- Permite que você carregue um arquivo de máscaras (extensão .dem) e armazene seus dados na lista de máscaras disponíveis selecionando o botão Masks. Experimente criar o seu próprio arquivo de máscaras, disponibilizando-as através da caixa de diálogo Input Mask Editor.

Para definir uma máscara para o campo CEP, atribua o valor 00000\-999;0; à propriedade EditMask do objeto ClientDataSetSociosCEP.

Esta máscara permite que se digitem todos os caracteres ou apenas os cinco primeiros.

Outro campo que deve ter máscara é o campo CPF. Defina da seguinte forma a máscara deste campo:

- Atribua o valor 000.000.000-00;0; à propriedade EditMask do objeto ClientDatasetSociosCPF.

Experimente acrescentar um novo registro e verifique que os campos para os quais foram definidas máscaras não permitem a digitação de caracteres inválidos.

O Componente MaskEdit

Em situações em que você precisa definir uma máscara para um texto a ser digitado pelo usuário, mas este texto não será armazenado em um campo de um registro, é possível usar o componente MaskEdit, que também possui a propriedade EditMask.

O componente MaskEdit está localizado na página Additional da paleta de componentes.

Fazendo a Tecla Enter Funcionar Como Tab

Os usuários que necessitam digitar uma grande quantidade de dados geralmente preferem navegar pelos diversos controles utilizando a tecla Enter em vez da tecla Tab.

Para incorporar essa funcionalidade ao nosso aplicativo, devemos definir um procedimento associado ao evento OnKeyPress dos diversos controles com os quais o usuário deverá interagir.

Esse procedimento normalmente recebe, como um dos seus parâmetros, uma variável chamada Var (passada por referência) que armazena o código da tecla pressionada.

Para verificar se a tecla pressionada foi a tecla Enter, basta verificar se o valor armazenado nessa variável é igual a #13. Em caso positivo, deve-se atribuir o valor #0 a essa variável (de forma a evitar que se escute um bip desagradável) e utilizar o procedimento SelectNext para transferir o foco para o próximo controle (próximo de acordo com o valor da propriedade TabOrder).

Este procedimento recebe como parâmetros:

- O controle atual.
- Um valor booleano, indicando se o próximo controle será o que possui o valor da sua propriedade TabOrder imediatamente superior ao do controle atual (true) ou imediatamente inferior ao do controle atual (false).
- Um valor booleano, indicando se o próximo controle deverá ou não ter o valor da sua propriedade TabStop igual a True.

Dessa maneira, basta codificar da seguinte forma o procedimento associado ao evento OnKeyPress do componente DBEditCodigoSocio, procedimento este que deverá ser compartilhado pelos demais componentes DBEdit do formulário.

```
procedure TFormCadastraSocios.DBEditCodigoClienteKeyPress(
  Sender: TObject; var Key: Char);
begin
     if (key = #13) then
     begin
         Key := #0;
         SelectNext((Sender as TWinControl),True,True);
     end;
end;
```

Compartilhe este evento também com o componente BotaoCadastrar e defina como False a propriedade TabStop do componente BotaoFechar.

Não esqueça de configurar adequadamente a propriedade TabStop dos diversos componentes.

O Componente DBCombobox

Nosso formulário já está funcionando corretamente, mas poderemos aperfeiçoá-lo ainda mais se considerarmos que, ao invés de exigir do usuário que digite um valor para o campo Estado, podemos dar a ele a possibilidade de selecionar este valor a partir de uma lista contendo todos os estados da federação – o que pode ser feito utilizando-se o componente DBCombobox, situado na página Data Controls da paleta de componentes.

Propriedades do Controle DBComboBox

O componente DBCombobox, da mesma forma que o componente DBEdit, possui as mesmas propriedades Datasource e Datafield que definem a tabela e o campo ao qual está vinculado. Entre as outras principais propriedades deste componente, podem-se destacar:

- ItemIndex: Define o índice da string selecionada da lista. O primeiro item possui o índice 0, o segundo possui o índice 1, etc., até o n-ésimo item, que possui índice n-1.
- Left e Top: Definem a posição do componente no formulário.
- Name: Esta propriedade define o nome pelo qual o objeto será referenciado no código da aplicação.
- Height e Width: Definem as dimensões do componente.
- Style: Define o estilo do componente. Esta propriedade pode assumir os valores descritos na tabela a seguir:

Valor	Significado
TACsDropDown:	Cria uma lista drop-down com uma caixa de edição na qual o usuário pode digitar texto.
TAcsSimple:	Cria uma caixa de edição sem uma lista drop-down.

Valor	Significado
TAcsDropDownList:	Cria uma lista drop-down sem uma caixa de edição.
TAcsOwnerDrawFixed:	Cria uma lista drop-down sem caixa de edição em que os itens podem ser qualquer objeto definido pelo usuário e não apenas strings, sendo que todos eles têm altura fixa.
TAcsOwnerDrawVariable:	Cria uma lista drop-down sem caixa de edição em que os itens podem ser qualquer objeto definido pelo usuário e não apenas strings, sendo que eles não têm altura fixa.

- Text: Define o texto que está sendo exibido na caixa de texto do componente.
- Items: Esta propriedade consiste em uma lista de strings, que associa uma linha de texto a cada um dos itens da lista. Esta propriedade pode ser alterada mediante a inclusão de um trecho de código ou diretamente no Object Inspector.

Propriedades do Objeto Items (da Classe TStrings)

Entre as propriedades do objeto Items, podem-se destacar:

- Sorted: Esta propriedade é declarada como uma variável booleana e define se os itens estão ordenados alfabeticamente.
- Count: Esta propriedade é declarada como uma variável inteira e define o número de strings na lista.

Métodos do Objeto Items (da classe TStrings)

Entre os métodos do objeto Items, podem-se destacar:

- Add: Adiciona uma string à lista.

Para adicionar a string "exemplo" na lista de strings exibida por um componente do tipo DBComboBox chamado DBComboBox1, basta incluir uma linha de código como:

```
DBComboBox1.Items.Add('exemplo');
```

- Clear: Deleta todas as strings da lista.

Para deletar todas as strings na lista de strings exibida por um componente do tipo DBComboBox chamado DBComboBox1, basta incluir uma linha de código como:

```
DBComboBox1.Items.Clear;
```

- Delete: Remove uma string da lista.

Para remover a string selecionada na lista de strings exibida por um componente do tipo DBComboBox chamado DBComboBox1, basta incluir uma linha de código como:

```
DBComboBox1.Items.Delete(DBComboBox1.ItemIndex);
```

- Insert: Insere uma string na lista, em uma posição predeterminada.

Para inserir a string "exemplo" na primeira posição da lista de strings exibida por um componente do tipo DBComboBox chamado DBComboBox1, basta incluir uma linha de código como:

```
DBComboBox1.Items.Insert(0,'exemplo');
```

- LoadFromFile: Carrega a lista de strings a ser armazenada a partir de um arquivo.

Para colocar o texto armazenado no arquivo dados.dat na lista de strings exibida por um componente do tipo DBComboBox chamado DBComboBox1, basta incluir uma linha de código como:

```
DBComboBox1.Items.LoadFromFile('dados.dat');
```

- SaveToFile: Salva a lista de strings em um arquivo.

Para colocar a lista de strings armazenada no componente em um arquivo chamado dados.dat, basta incluir uma linha de código como:

```
DBComboBox1.Items.SaveToFile('dados.dat');
```

A propriedade chamada Items – na realidade, uma lista de strings (objeto da classe TStrings) – é que permite armazenar uma relação de valores que podem ser selecionados pelo usuário. Neste caso, queremos armazenar nesta lista de strings as siglas de cada um dos 27 estados da federação, relacionadas a seguir:

```
AC
AL
AM
AP
BA
CE
DF
ES
GO
MA
MG
MT
MS
PA
PB
PE
PI
PR
RJ
RN
RR
RO
RS
SC
SE
SP
TO
```

Para substituir o componente DBEditEstadoSocio pelo componente DBCombobox, você deverá executar os seguintes procedimentos:

1. Selecionar o componente DBEditEstadoSocio e pressionar a tecla Del, para removê-lo.
2. Incluir um componente DBCombobox (o oitavo componente da página Data Controls) no local anteriormente ocupado pelo componente DBEditEstadoSocio.

3. Alterar a propriedade Name deste componente para DBComboboxEstadoSocio.
4. Selecionar as reticências exibidas à direita da propriedade Items deste componente para exibir a caixa de diálogo String List Editor e digitar as siglas dos 27 estados.
5. Alterar o valor da propriedade Style deste componente para csDropDownList.
6. Utilizando a caixa de diálogo Edit Tab Order (mostrada na figura a seguir e exibida selecionando-se o item Tab Order no menu pop-up do formulário), ordenar os componentes como indicado.

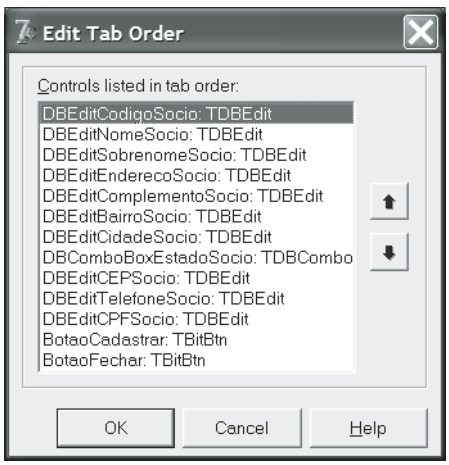


Figura 10.20: A caixa de diálogo Edit Tab Order.

7. Compartilhar com este componente o procedimento associado ao evento OnKeyPress do componente DBEditCodigoSocio (que já está sendo compartilhado pelos demais componentes). Desta maneira, a tecla Enter funcionará como Tab também para este componente.

Destacando o Componente Que Recebe o Foco

Para que o usuário não se perca durante a digitação de informações, seria interessante destacar o componente que possui o foco da aplicação.

Isto pode ser feito codificando-se adequadamente os procedimentos associados ao evento OnEnter e OnExit do componente. O evento OnEnter ocorre sempre que o componente recebe o foco (nada tem a ver com a tecla Enter, como pode ser sugerido pelo seu nome). Já o evento OnExit ocorre sempre que o componente perde o foco.

Desta maneira, seria interessante alterarmos a cor do componente quando o mesmo recebe o foco, e devolver a sua cor original quando o foco sai do componente. Isto pode ser feito definindo-se da seguinte maneira os procedimentos associados aos Eventos OnEnter e OnExit do componente DBEditCodigoCliente (a serem compartilhados pelos demais componentes DBEdit). No componente DBCombobox não precisa, pois fará isso automaticamente, e se tentar tratá-lo com um DBEdit, provocará um erro.

```
procedure TFormCadastraSocios.DBEditCodigoSocioEnter(Sender: TObject);
begin
    (Sender as TDBEdit).Color := clActiveCaption;
```

```
    (Sender as TDBEdit).Font.Color := clCaptionText;
end;

procedure TFormCadastraSocios.DBEditCodigoSocioExit(Sender: TObject);
begin
    (Sender as TDBEdit).Color := clWindow;
    (Sender as TDBEdit).Font.Color := clWindowText;
end;
```

Gostaria de, neste ponto, fazer uma observação importante. Você já deve ter reparado que, na maioria dos procedimentos associados a eventos, existe um parâmetro denominado Sender. Este parâmetro é muito útil quando compartilhamos procedimentos associados a eventos entre diversos componentes, pois identifica qual destes componentes disparou o evento – o que é muito importante quando precisamos alterar as propriedades do componente que disparou o evento (pois precisamos identificá-lo). Como o parâmetro Sender é definido como sendo um objeto da classe TObject, precisamos tratá-lo como sendo um objeto da classe do componente que disparou o evento – o que pode ser feito usando-se o operador de conversão As.

Pronto! Agora nosso formulário de Cadastro de Clientes está mais interessante e funcional, e deve apresentar o aspecto mostrado na figura a seguir.

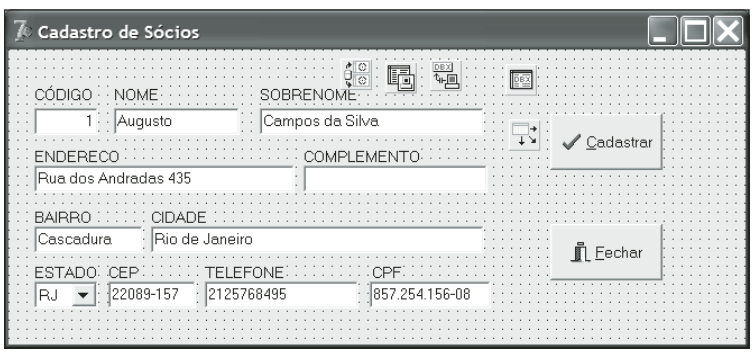


Figura 10.21: Aspecto final do formulário de Cadastro de Sócios.

Capítulo 11

Criação de um Repositório Para Componentes de Acesso a Dados

Neste capítulo será apresentado o Data Module, um repositório de objetos destinado a armazenar componentes para acesso a bancos de dados, e as vantagens da sua utilização.

CRIAÇÃO DE DATA MODULES

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Conhecimentos básicos sobre a linguagem Object Pascal (já apresentados nos capítulos anteriores) e os componentes básicos de acesso a dados da página DBExpress da paleta de componentes.
- Noções Básicas sobre bancos de dados.

METODOLOGIA

- Apresentação e descrição dos principais conceitos relacionados à criação e utilização de objetos Data Module.

TÉCNICA

- Apresentação e descrição dos procedimentos necessários à criação de Data Modules para armazenamento de componentes de acesso a dados.

OS OBJETOS DO TIPO DATAMODULE

No capítulo anterior foram apresentados os procedimentos necessários à criação de um formulário no qual foram inseridos os componentes responsáveis por acessar a tabela de dados dos Sócios. Esta abordagem apresenta a desvantagem de que, caso vários formulários acessem uma mesma tabela, todos deverão conter estes componentes de acesso. Neste caso, uma boa alternativa consiste em empregar um Data Module.

Os objetos do tipo DataModule servem como um repositório para o armazenamento de componentes não-visuais de acesso a bancos de dados, permitindo que estes sejam acessados pelo formulário da aplicação, centralizando o acesso a dados por parte da aplicação.

Para criar um Data Module, você deve executar os seguintes procedimentos:

1. Selecionar o item New/Data Module do menu File do Delphi 7.

Ou:

1. Selecionar o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items, selecionar o item correspondente ao Data Module e o botão Ok.

Será então criado um Data Module, como mostrado na figura a seguir.

2. Altere a propriedade Name deste DataModule para Dados.
3. Selecione o item Save as, do menu File, e salve a unidade de código associada a este DataModule com o nome UnitDados.pas.

Embora o objeto DataModule não seja um formulário comum, para exibi-lo devem ser executados os mesmos procedimentos usados para exibir qualquer formulário.

Figura 11.1: O objeto DataModule.

4. Exiba o formulário de Cadastro de Sócios e selecione os diversos componentes de acesso, como mostrado na figura a seguir.

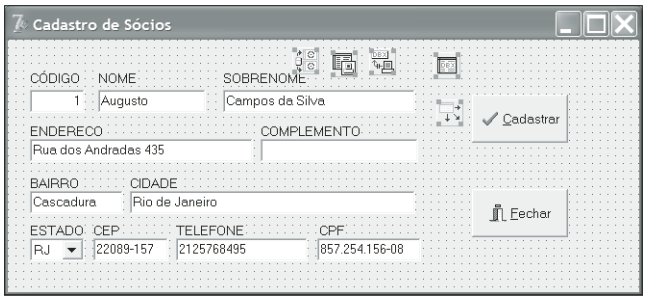


Figura 11.2: Selecionando os componentes de acesso a dados no formulário de Cadastro de Sócios.

5. Selecione o item Cut do menu Edit do ambiente de desenvolvimento do Delphi 7.
6. Exiba novamente o Datamodule.
7. Selecione o item Paste do menu Edit do ambiente de desenvolvimento do Delphi 7. Os componentes de acesso serão incluídos no formulário, como mostrado na figura a seguir.

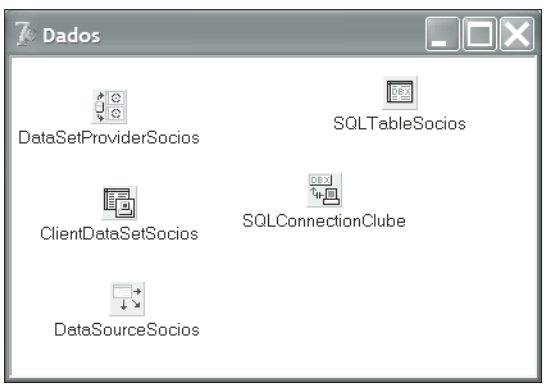


Figura 11.3: Copiando os componentes de acesso a dados do formulário de cadastro de sócios para o Datamodule.

Caso seja necessário, reposicione os diversos componentes no Datamodule.

8. Exiba novamente o formulário de Cadastro de Sócios e selecione o item Use Unit do menu File do ambiente de desenvolvimento do Delphi 7. Será exibida a caixa de diálogo Use Unit, mostrada na figura a seguir.

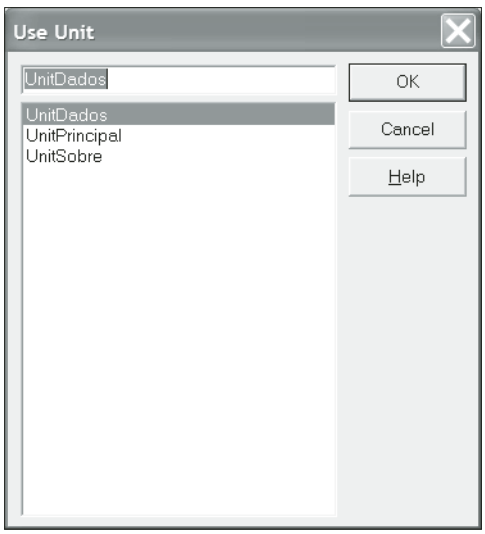


Figura 11.4: A caixa de diálogo Use Unit.

9. Nesta caixa de diálogo, selecione a unit UnitDados e o botão Ok.
10. Selecione cada um dos componentes DBEdit e o DBCombobox e altere a propriedade Datasource de cada um deles para Dados.DatasourceSocios.
11. Na unit UnitCadastraSocios, troque cada referência a ClientDatasetSocios para Dados.ClientDatasetSocios. Isto se deve ao fato de que, a partir de agora, o componente ClientDataSocios é um campo interno ao objeto Dados, e daí a necessidade da sua qualificação. Desta maneira, os procedimentos associados a eventos na unit UnitCadastraSocios passarão a ter a seguinte codificação:

```
procedure TFormCadastraSocios.FormShow(Sender: TObject);
begin
    DBEditCodigoSocio.SetFocus;
    Dados.ClientDatasetSocios.Open;
    Dados.ClientDatasetSocios.Append;
end;

procedure TFormCadastraSocios.BotaoCadastrarClick(Sender: TObject);
begin
    Dados.ClientDatasetSocios.Post;
    Dados.ClientDatasetSocios.Append;
    DBEditCodigoSocio.SetFocus;
```

```
end;

procedure TFormCadastraSocios.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    Dados.ClientDatasetSocios.Delete;
    Dados.ClientDatasetSocios.ApplyUpdates(0);
end;

procedure TFormCadastraSocios.DBEditCodigoSocioKeyPress(Sender: TObject;
  var Key: Char);
begin
     if (key = #13) then
     begin
         Key := #0;
         SelectNext((Sender as TWinControl),True,True);
    end;
end;

procedure TFormCadastraSocios.DBEditCodigoSocioEnter(Sender: TObject);
begin
    (Sender as TDBEdit).Color := clActiveCaption;
    (Sender as TDBEdit).Font.Color := clCaptionText;
end;

procedure TFormCadastraSocios.DBEditCodigoSocioExit(Sender: TObject);
begin
    (Sender as TDBEdit).Color := clWindow;
    (Sender as TDBEdit).Font.Color := clWindowText;
end;
```

Para exibir o objeto DataModule criado nos tópicos anteriores, execute os seguintes procedimentos:

1. Selecione a opção Forms no menu View.

Será apresentada a lista de formulários disponíveis (Figura 11.5).

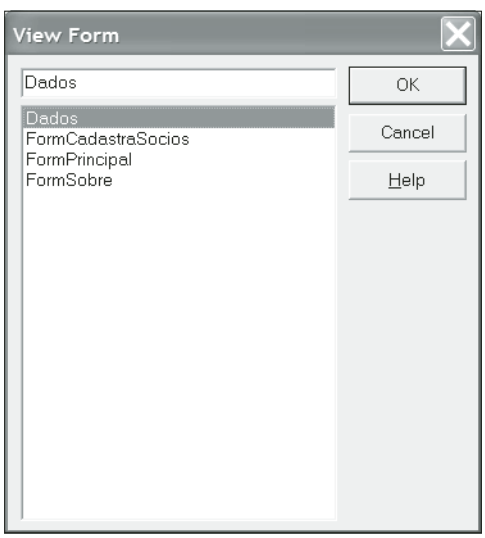


Figura 11.5: Selecionando um formulário ou objeto DataModule.

2. Selecione Dados e depois o botão OK.

Será exibido o objeto Dados (o DataModule).

Os Objetos de Acesso ao Banco de Dados

Observando novamente a Figura 11.3, verificamos que os seguintes componentes foram inseridos no objeto DataModule:

- Um componente que representa a conexão ao banco de dados. Este componente é denominado SQLConnection, e foi abordado anteriormente.
- Um componente que representa uma tabela de um banco de dados. Este componente é denominado SQLTable, e só permite acesso unidirecional aos registros da tabela.
- Um componente que faz a ligação entre o componente SQLTable e o componente ClientDataset, que permitirá o acesso bidirecional aos registros da tabela. Este componente é denominado DatasetProvider. Este componente se conecta ao componente SQLTable através da sua propriedade Dataset.
- Um componente ClientDataset, que como já foi dito permitirá o acesso bidirecional aos registros da tabela, e permitindo a sua edição local (na memória), a ser posteriormente efetivada no banco de dados, mediante uma chamada ao seu método ApplyUpdates. Este componente se conecta ao componente DatasetProvider através da sua propriedade ProviderName.
- Um componente que estabelece a conexão entre o componente que representa uma tabela de um banco de dados e os componentes destinados à sua visualização. Este componente é denominado DataSource, e se conecta ao componente SQLTable através da sua propriedade Dataset.

O Componente ClientDataset

Conforme visto no tópico anterior, o componente ClientDataset representa uma tabela de um banco de dados, permitindo a sua edição local e com acesso bidirecional, o que não é permitido pelo componente SQLTable. Este componente possui um "cursor" ou "ponteiro" que indica o registro corrente da tabela na memória, isto é, aquele registro com o qual se está trabalhando localmente no momento.

Principais Propriedades do Componente ClientDataset

Entre as principais propriedades de um componente do tipo ClientDataset, podemos destacar:

- Active: Esta propriedade é definida como uma variável booleana, e define se uma tabela está ou não ativa na memória. Definir Active como True ou False equivale a chamar os seus métodos Open e Close, respectivamente (estes métodos serão descritos a seguir).
- BOF: Esta propriedade é declarada como uma variável booleana. Quando o seu valor for igual a True, indica que o cursor está posicionado no primeiro registro da tabela na memória.
- ProviderName: Esta propriedade serve para informar o nome do componente DatasetProvider ao qual pertence está conectado.
- EOF: Esta propriedade é declarada como uma variável booleana. Quando o seu valor for igual a True, indica que o cursor está posicionado no últmo registro da tabela na memória.

- Fields: Esta propriedade é um array que armazena, em cada um dos seus elementos, o valor armazenado em cada um dos campos do registro corrente da tabela representada pelo componente. O valor do primeiro campo está armazenado em Fields[0], o do segundo campo em Fields[1], etc., de maneira que o valor armazenado no n-ésimo campo está em Fields[n-1].
- IndexName: Esta propriedade armazena o nome do índice corrente da tabela na memória.
- Name: Esta propriedade, como em qualquer objeto, define o nome pelo qual o componente será referenciado no programa.
- RecordCount: Esta propriedade armazena o número de registros existentes na tabela.

Principais Métodos do Componente ClientDataset

Além das propriedades descritas no tópico anterior, este componente também tem métodos que nos permitem manipular os registros de uma tabela na memória. Entre estes métodos, podemos destacar:

- Append: Este método acrescenta um registro em branco no final da tabela na memória.
- Cancel: Este método desfaz as alterações feitas no registro corrente (desde que estas alterações ainda não tenham sido gravadas, ainda que localmente na memória).
- Close: Este método bloqueia o acesso a uma tabela na memória (equivale a definir o valor da propriedade Active como False).
- Delete: Este método deleta (remove) o registro corrente da tabela na memória.
- Edit: Este método permite a edição dos campos do registro corrente na memória.
- FieldByName: Este método retorna o valor armazenado em um campo do registro corrente na memória, cujo nome é passado como parâmetro na forma de uma string. Por exemplo, para obter o valor armazenado no campo Nome do registro corrente da tabela Sócios na memória, que é representado pelo componente ClientDatasetSocios, e atribuí-lo a uma variável "s" do tipo string, basta incluir a seguinte linha de código:

```
s:= ClientDatasetSocios.FieldByName('Nome').AsString;
```

- First: Este método coloca o primeiro registro da tabela na memória como o registro corrente (move o "cursor" para o primeiro registro da tabela na memória).
- Last: Este método coloca o último registro da tabela na memória como o registro corrente (move o "cursor" para o primeiro registro da tabela na memória).
- Next: Este método coloca o próximo registro da tabela na memória como o registro corrente (move o "cursor" para o próximo registro da tabela na memória).
- Open: Este método abre uma tabela na memória, tornando-a ativa e fornecendo acesso aos seus registros (equivale a definir o valor da propriedade Active como True).
- Post: Este método realiza a gravação do registro corrente na memória.
- Prior: Este método coloca o registro anterior da tabela na memória como o registro corrente (move o "cursor" para o registro anterior da tabela na memória).
- Refresh: Este método atualiza os dados exibidos pelos registros da tabela na memória.

Existem outros métodos, usados para aplicar filtros a uma tabela e para realizar pesquisas em registros, que serão vistos posteriormente.

A ordem dos registros é definida pelos campo(s) que forma(m) o índice corrente da tabela.

O Componente DataSource

Este componente é de importância fundamental no desenvolvimento de aplicativos de bancos de dados com o Delphi, pois é ele quem se encarrega de fazer a conexão entre os componentes de visualização (a serem vistos posteriormente) e os componentes de acesso (no nosso caso, as tabelas que compõem o banco de dados).

Exemplificando: para acessar a tabela Sócios, representada na memória por um componente do tipo ClientDatasetSocios, precisamos incluir um componente DataSource para realizar a conexão entre a tabela na memória e os componentes que exibirão os valores armazenados nos campos dos seus registros (os componentes de visualização). Os componentes de visualização que desejarem acessar esta tabela na memória não o farão diretamente, mas através do componente DataSource, que serve como uma "ponte" entre estes e as tabelas.

Principais Propriedades do Componente DataSource

Entre as principais propriedades de um componente do tipo DataSource, podemos destacar:

- DataSet: Esta propriedade é usada para definir o componente cuja conexão desejamos estabelecer.
- Name: Esta propriedade, como em qualquer objeto, define o nome pelo qual o componente será referenciado no programa.

Desta maneira, qualquer componente que queira acessar os dados da tabela Sócios na memória, que é representada por SQLCientDatasetSocios, deve fazê-lo através do componente DataSourceSocios (o componente Datasource empregado neste exemplo).

Fica então estabelecida a seguinte regra: para cada tabela que quisermos acessar, representada por um componente do tipo ClientDataset (ou qualquer outro que represente uma tabela, como o SQLTable, Table – no caso do BDE, ADOTable – no caso do ADO, IBTable – no caso do IBX, etc.), devemos incluir um componente do tipo DataSource cuja propriedade DataSet é igual ao valor da propriedade Nome do componente que representa a tabela.

O Componente SimpleDataset

Este componente foi incorporado a esta versão do Delphi (substituindo o componente SQLClientDataset incorporado na versão 6), e tem por finalidade substituir os componentes SQLConnection, SQLTable, DatasetProvider e ClientDataset. Na verdade, ele incorpora um componente SQLDataset, que será visto posteriormente. Nestes exemplos, não eliminaremos a necessidade de uso do componente SQLConnection (no caso o componente SQLConnectionClube), pois vários componentes usarão a conexão definida por este componente.

Este componente é o último da página DBExpress da paleta de componentes, e para utilizá-lo para acessar a tabela Sócios, execute os seguintes procedimentos:

1. Exiba o Datamodule Dados.
2. Selecione simultaneamente os componentes SQLTableSocios, DatasetProviderSocios e ClientDatasetSocios.
3. Selecione a tecla Del, para remover os componentes selecionados.
4. Insira um componente SimpleDataset no Datamodule e configure da seguinte maneira as suas principais propriedades:

   Name: SimpleDatasetSocios.
   Connection: SQLConnectionClube.
   Subpropriedade CommandType da propriedade Dataset: ctTable.
   Subpropriedade Command da propriedade Dataset: SOCIOS.
   Subpropriedade Active da propriedade Dataset: True.

5. Altere a propriedade Dataset do componente DatasourceSocios para SimpleDatasetSocios.
6. Redefina da seguinte maneira os procedimentos associados a eventos da unit UnitCadastraSocios que referenciavam o componente ClientDatasetSocios:

```
procedure TFormCadastraSocios.FormShow(Sender: TObject);
begin
    DBEditCodigoSocio.SetFocus;
    Dados.SimpleDatasetSocios.Open;
    Dados.SimpleDatasetSocios.Append;
end;

procedure TFormCadastraSocios.BotaoCadastrarClick(Sender: TObject);
begin
    Dados.SimpleDatasetSocios.Post;
    Dados.SimpleDatasetSocios.Append;
    DBEditCodigoSocio.SetFocus;
end;

procedure TFormCadastraSocios.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    Dados.SimpleDatasetSocios.Delete;
    Dados.SimpleDatasetSocios.ApplyUpdates(0);
end;
```

7. Usando o Fields Editor, selecione o item Add All Fields de seu menu pop-up para adicionar objetos para representar todos os campos desta tabela.
6. Redefina a propriedade EditMask para os objetos que representam os campos CEP e CPF, como descrito no capítulo anterior.

Inserindo os Demais Componentes de Acesso

Para que as demais tabelas possam ser acessadas pelo nosso aplicativo, devemos incluir no Datamodule os componentes SimpleDataset e DataSource necessários. Para isso, basta executar os seguintes procedimentos:

Para a tabela de Atividades:

1. Inclua um componente SimpleDataset e atribua os seguintes valores às suas principais propriedades:

   Name: SimpleDatasetAtividades.
   Connection: SQLConnectionClube.
   Subpropriedade CommandType da propriedade Dataset: ctTable.
   Subpropriedade Command da propriedade Dataset: ATIVIDADES.
   Subpropriedade Active da propriedade Dataset: True.

2. Inclua um componente DataSource e atribua os seguintes valores às suas principais propriedades:

   Name: DatasourceAtividades.
   Dataset: SimpleDatasetAtividades.

3. Usando o Fields Editor, selecione o item Add All Fields de seu menu pop-up para adicionar objetos para representar todos os campos desta tabela.

4. Atribua o valor ########.00 às propriedades EditFormat e DisplayFormat do objeto que representa o campo VALOR desta tabela. Isto fará com que o número seja sempre exibido com duas casas decimais (estas propriedades definem a "máscara"de campos numéricos).

Para a tabela de Matrículas:

1. Inclua um componente SimpleDataset e atribua os seguintes valores às suas principais propriedades:

   Name: SimpleDatasetMatriculas.
   Connection: SQLConnectionClube.
   Subpropriedade CommandType da propriedade Dataset: ctTable.
   Subpropriedade Command da propriedade Dataset: MATRICULAS.
   Subpropriedade Active da propriedade Dataset: True.

2. Inclua um componente DataSource e atribua os seguintes valores às suas principais propriedades:

   Name: DatasourceMatriculas.
   Dataset: SimpleDatasetMatriculas.

3. Usando o Fields Editor, selecione o item Add All Fields de seu menu pop-up para adicionar objetos para representar todos os campos desta tabela.

Após a inclusão destes componentes, seu Datamodule deverá ficar com o aspecto mostrado na Figura a seguir:

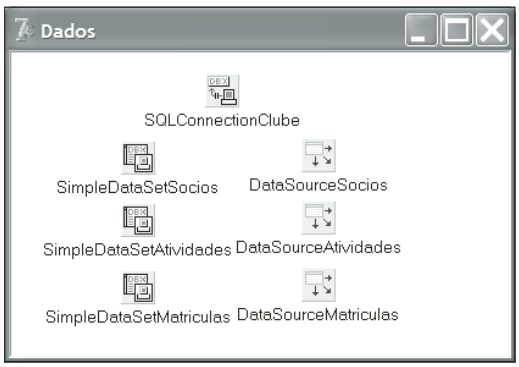


Figura 11.6: Aspecto do objeto Datamodule após a inclusão dos demais componentes de acesso.

Se necessário, redimensione o Datamodule para que todos os componentes possam ser visualizados simultaneamente.

Definindo Índices nos Componentes de Acesso

O componente SimpleDataset permite a definição de índices locais, que são conjuntos de dois ou mais campos pelos quais os dados podem ser ordenados. Um índice pode ser simples (quando é formado por um único campo) ou composto (quando é formado por dois ou mais campos).

Neste tópico serão apresentados os procedimentos necessários à criação de índices simples e compostos.

Criando um Índice Simples

Quando consultarmos a tabela de atividades, será interessante que estes dados estejam ordenados alfabeticamente pelos nomes das atividades. Neste caso, devemos criar um índice simples, baseado no campo Nome.

Para criar este índice, você deve executar os seguintes procedimentos:

1. Selecione o componente SimpleDatasetAtividades.
2. No Object Inspector, selecione os três pontinhos à direita da propriedade IndexDefs, para exibir a janela de edição de índices, mostrada na figura a seguir.

Figura 11.7: A janela de edição de índices.

3. Selecione o botão direito do mouse para exibir o menu pop-up desta janela e, neste menu, selecione o índice Add, para criar um novo índice na memória. Este índice será representado por um objeto da classe TIndexDefs, como mostrado na figura a seguir.

Figura 11.8: Criando um objeto da classe TIndexDefs.

4. Altere a propriedade Name deste objeto para indNome, usando o Object Inspector.
5. Defina como NOME a propriedade Fields deste objeto no Object Inspector. Isto indica que este será um índice simples, formado pelo campo Nome.

6. Altere para indNome a propriedade indexName do componente SimpleDatasetAtividades, usando o Object Inspector.

Pronto! Seu índice simples foi criado!

Criando um Índice Composto

Quando consultarmos a tabela de sócios, será interessante que estes dados estejam ordenados alfabeticamente pelos seus nomes. Neste caso, devemos criar um índice composto, baseado no campo Nome e Sobrenome.

Para criar este índice, você deve executar os seguintes procedimentos:

1. Selecione o componente SimpleDatasetSocios.
2. No Object Inspector, selecione os três pontinhos à direita da propriedade IndexDefs, para exibir a janela de edição de índices.
3. Selecione o botão direito do mouse para exibir o menu pop-up desta janela e, neste menu, selecione o índice Add, para criar um novo índice na memória. Este índice será representado por um objeto da classe TIndexDefs.
4. Altere a propriedade Name deste objeto para indNome, usando o Object Inspector.
5. Defina como NOME;SOBRENOME (Os dois campos separados por um ponto-e-vírgula) a propriedade Fields deste objeto no Object Inspector. Isto indica que este será um índice composto, formado pelos campos Nome e Sobrenome.
6. Altere para indNome a propriedade indexName do componente SimpleDatasetSocios, usando o Object Inspector.

Pronto! Seu índice composto foi criado!

Criando Chaves Primárias

Existem situações em que dois registros não podem ter o mesmo valor em um determinado campo (ou grupo de campos), e neste caso dizemos que este campo (ou estes campos) formam uma chave primária simples (composta).

No caso das tabelas de Sócios e de Atividades, por exemplo, o campo referente a seu código deve ser único. Ou seja, devemos ter uma chave primária definida pelo campo CodigoSocio na tabela de sócios e CodigoAtividade na tabela de Atividades.

No caso da tabela de matrículas, devemos ter uma chave primária composta formada pelos campos Sócio e Atividade (de modo a evitar que um mesmo sócio seja matriculado duas vezes em uma mesma atividade).

Criando uma Chave Primária na Tabela de Sócios

Para criar uma chave primária formada pelo campo CodigoSocio na tabela de Sócios, você deve executar os seguintes procedimentos:

1. Selecione o componente SimpleDatasetSocios.
2. No Object Inspector, selecione os três pontinhos à direita da propriedade IndexDefs, para exibir a janela de edição de índices.
3. Selecione o botão direito do mouse para exibir o menu pop-up desta janela e, neste menu, selecione o índice Add, para criar um novo índice na memória. Este índice será representado por um objeto da classe TIndexDefs.
4. Altere a propriedade Name deste objeto para indCodigoSocio, usando o Object Inspector.
5. Defina como CODIGOSOCIO a propriedade Fields deste objeto no Object Inspector.
6. Altere para True o valor da subpropriedade ixPrimary da propriedade Options deste objeto, usando o Object Inspector.
7. Altere para True o valor da subpropriedade ixUnique da propriedade Options deste objeto, usando o Object Inspector.
8. Redefina da seguinte maneira o procedimento associado ao evento OnShow do formulário:

```
procedure TFormCadastraSocios.FormShow(Sender: TObject);
begin
    Dados.SimpleDatasetSocios.IndexName := 'indCodigoSocio';
    DBEditCodigoSocio.SetFocus;
    Dados.SimpleDatasetSocios.Open;
    Dados.SimpleDatasetSocios.Append;
end;
```

Desta maneira, o índice será trocado ao se exibir o formulário.

9. Redefina da seguinte maneira o procedimento associado ao evento OnClose do formulário:

```
procedure TFormCadastraSocios.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    Dados.SimpleDatasetSocios.Delete;
    Dados.SimpleDatasetSocios.ApplyUpdates(0);
    Dados.SimpleDatasetSocios.IndexName := 'indNomeCompleto';
end;
```

10. Redefina da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoCadastrar:

```
procedure TFormCadastraSocios.BotaoCadastrarClick(Sender: TObject);
begin
  try
    Dados.SimpleDatasetSocios.Post;
    Dados.SimpleDatasetSocios.Append;
  except
    ShowMessage("Este Código Já Foi Cadastrado");
  end;
    DBEditCodigoSocio.SetFocus;
end;
```

Desta maneira, o índice será novamente trocado ao se fechar o formulário.

Criando uma Chave Primária na Tabela de Atividades

Para criar uma chave primária formada pelo campo CodigoAtividade na tabela de Atividades, você deve executar os seguintes procedimentos:

1. Selecione o componente SimpleDatasetAtividade.
2. No Object Inspector, selecione os três pontinhos à direita da propriedade IndexDefs, para exibir a janela de edição de índices.
3. Selecione o botão direito do mouse para exibir o menu pop-up desta janela e, neste menu, selecione o índice Add, para criar um novo índice na memória. Este índice será representado por um objeto da classe TIndexDefs.
4. Altere a propriedade Name deste objeto para indCodigoAtividade, usando o Object Inspector.
5. Defina como CODIGOATIVIDADE a propriedade Fields deste objeto no Object Inspector.
6. Altere para True o valor da subpropriedade ixPrimary da propriedade Options deste objeto, usando o Object Inspector.
7. Altere para True o valor da subpropriedade ixUnique da propriedade Options deste objeto, usando o Object Inspector.

Criando uma Chave Primária Composta na Tabela de Matrículas

Para criar uma chave primária composta formada pelos campos SOCIO e ATIVIDADE na tabela de Matriculas, você deve executar os seguintes procedimentos:

1. Selecione o componente SimpleDatasetMatricula.
2. No Object Inspector, selecione os três pontinhos à direita da propriedade IndexDefs, para exibir a janela de edição de índices.
3. Selecione o botão direito do mouse para exibir o menu pop-up desta janela e, neste menu, selecione o índice Add, para criar um novo índice na memória. Este índice será representado por um objeto da classe TIndexDefs.
4. Altere a propriedade Name deste objeto para indMatricula, usando o Object Inspector.
5. Defina como SOCIO:ATIVIDADE a propriedade Fields deste objeto no Object Inspector.
6. Altere para True o valor da subpropriedade ixPrimary da propriedade Options deste objeto, usando o Object Inspector.
7. Altere para True o valor da subpropriedade ixUnique da propriedade Options deste objeto, usando o Object Inspector.

Capítulo 12

Criação de Formulários Para Cadastro de Fornecedores, Produtos e Pedidos

Neste capítulo serão apresentados os procedimentos necessários à criação de formulários de cadastro de atividades e matrículas.

Criação de Formulários de Cadastro

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Conhecimentos básicos sobre a linguagem Object Pascal (já apresentados nos capítulos anteriores).
- Criação de formulários e utilização dos componentes básicos de interface.
- Utilização de componentes para acesso a bancos de dados.

METODOLOGIA

- Apresentação e descrição dos principais conceitos relacionados à criação de formulários de cadastro.

TÉCNICA

- Apresentação e descrição dos procedimentos necessários à criação de formulários de cadastro.

Criando o Formulário de Cadastro de Atividades

Para criar um formulário de cadastro de atividades, execute os seguintes procedimentos:

1. Abra o projeto Clube.dpr, se ele já não estiver aberto no ambiente de desenvolvimento.
2. Crie um novo formulário, selecionando o item New/Form do menu File do Delphi 7.
3. Altere as propriedades Name e Caption deste formulário para FormCadastraAtividades e Cadastro de Atividades, respectivamente.
4. Salve a unit associada a este formulário com o nome UnitCadastraAtividades.
5. Selecione o componente SimpleDatasetAtividades do Datamodule, conforme indicado na Figura 12.1.

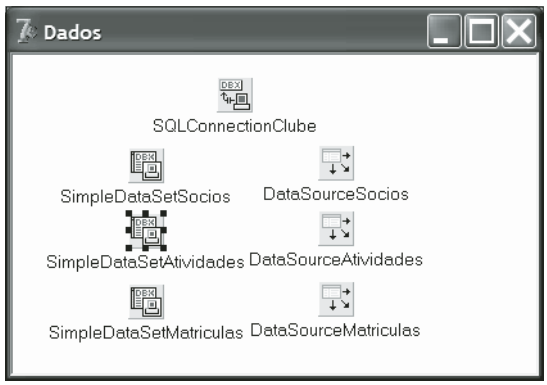


Figura 12.1: Selecionando o componente SimpleDatasetAtividades do Datamodule.

6. Pressione o botão direito do mouse com o componente SimpleDatasetAtividades selecionado, para exibir o seu menu pop-up.

7. Selecione o item Fields Editor deste menu pop-up. Será exibida a caixa de diálogo do Fields Editor.
8. Pressione o botão direito do mouse sobre a área branca do Fields Editor, para exibir o menu pop-up.
9. Selecione o item Add All Fields deste menu pop-up. Todos os campos serão exibidos na caixa de diálogo do Fields Editor, como mostrado na Figura 12.2.

Figura 12.2: Exibindo os campos da tabela no Fields Editor.

Conforme já descrito anteriormente, ao adicionar os campos da tabela no Fields Editor, o Delphi automaticamente cria objetos para representar estes campos.

10. Arraste os objetos do Fields Editor para o formulário FormCadastraAtividades para criar os componentes Labels e DBEdits vinculados aos campos da tabela, como mostrado na figura a seguir.

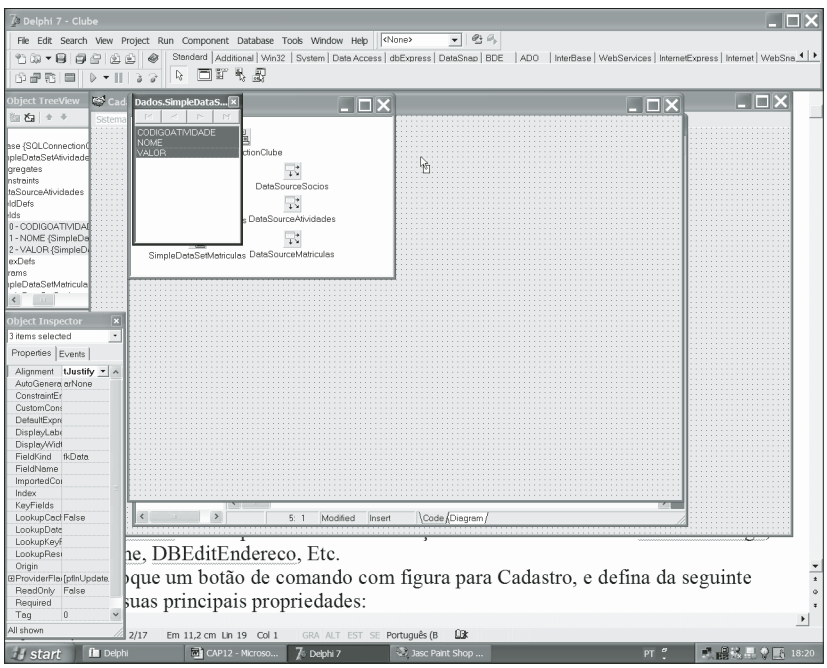


Figura 12.3: Arraste os objetos do Fields Editor para o formulário FormCadastraAtividades.

Ao liberar o botão esquerdo do mouse será exibida a mensagem mostrada na figura a seguir, informando que FormCadastraAtividades não inclui a unit Dados, e perguntando se deseja incluí-la. Selecione o botão Yes.

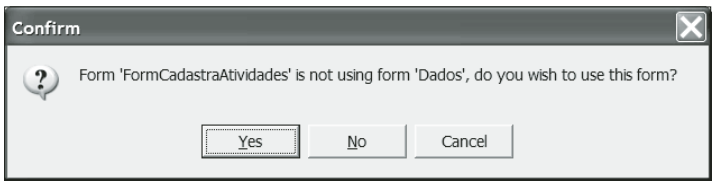


Figura 12.4: Mensagem de advertência.

Isto provoca a inclusão do nome da unit que contém o Datamodule Dados (UnitDados) na cláusula uses da unit UnitCadastraAtividades.

11. Altere a propriedade Caption do Label referente ao código da atividade para "CÓDIGO DA ATIVIDADE".
12. Renomeie os componentes de visualização dando-lhes os nomes DBEditCodigoAtividade, DBEditNomeAtividade e DBEditValorAtividade.
13. Coloque um botão de comando com figura (BitBtn) para Cadastro, e defina da seguinte maneira as suas principais propriedades:

    Kind: bkCustom.
    ModalResult: mrNone
    Name: BotaoCadastrar
    Caption: &Cadastrar
    Height: 50
    Width: 100

14. Para que estes dados sejam gravados na tabela quando o usuário selecionar este botão, devemos definir da seguinte maneira o procedimento associado ao evento OnClick deste botão.

```
procedure TFormCadastraAtividades.BotaoCadastrarClick(Sender: TObject);
begin
  try
    Dados.SimpleDatasetAtividades.Post;
    Dados.SimpleDatasetAtividades.Append;
  except
    ShowMessage("Este Código Já Foi Cadastrado");
  end;
    DBEditCodigoSocio.SetFocus;
end;
```

15. Defina da seguinte maneira o procedimento associado ao evento OnShow deste formulário:

```
procedure TFormCadastraAtividades.FormShow(Sender: TObject);
begin
    Dados.SimpleDatasetAtividades.IndexName := 'indCodigoSocio';
    DBEditCodigoSocio.SetFocus;
    Dados.SimpleDatasetAtividades.Open;
    Dados.SimpleDatasetAtividades.Append;
end;
```

Desta maneira, o índice será trocado ao se exibir o formulário.

16. Defina da seguinte maneira o procedimento associado ao evento OnKeyPress do componente DBEditCodigoAtividade, a ser compartilhado pelos demais componentes DBEdit:

```
procedure TFormCadastraAtividades.DBEditCodigoAtividadeKeyPress(
  Sender: TObject; var Key: Char);
begin
     if (key = #13) then
     begin
         Key := #0;
         SelectNext((Sender as TWinControl),True,True);
    end;
end;
```

17. Defina da seguinte maneira o procedimento associado ao evento OnClose deste formulário:

```
procedure TFormCadastraAtividades.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    Dados.SimpleDatasetAtividades.Delete;
    Dados.SimpleDatasetAtividades.ApplyUpdates(0);
    Dados.SimpleDatasetAtividades.IndexName := 'indNomeCompleto';
end;
```

Desta maneira, o índice será novamente trocado ao se fechar o formulário.

18. Defina da seguinte maneira o procedimento associado ao evento OnEnter do componente DBEditCodigoAtividade, a ser compartilhado pelos demais componentes DBEdit:

```
procedure TFormCadastraAtividades.DBEditCodigoAtividadeEnter(
  Sender: TObject);
begin
    (Sender as TDBEdit).Color := clActiveCaption;
    (Sender as TDBEdit).Font.Color := clCaptionText;
end;
```

19. Defina da seguinte maneira o procedimento associado ao evento OnExit do componente DBEditCodigoAtividade, a ser compartilhado pelos demais componentes DBEdit:

```
procedure TFormCadastraAtividades.DBEditCodigoAtividadeExit(
  Sender: TObject);
begin
    (Sender as TDBEdit).Color := clWindow;
    (Sender as TDBEdit).Font.Color := clWindowText;
end;
```

20. Inclua mais um botão de comando com figura e defina suas principais propriedades como indicado a seguir:

    Name: BotaoFechar
    Kind: bkClose
    Caption: &Fechar
    Height: 50
    Width: 100

Seu formulário deverá ficar com o aspecto mostrado na figura a seguir.

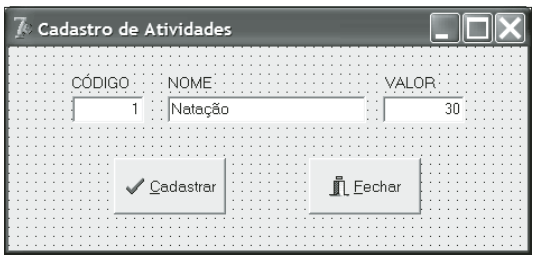


Figura 12.5: Aspecto final do formulário para Cadastro de Atividades.

21. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Cadastrar do menu Atividades, e compartilhe este procedimento com o evento OnClick do item Cadastrar do submenu Atividades do menu pop-up:

```
procedure TFormPrincipal.AtividadesCadastroClick(Sender: TObject);
begin
    FormCadastraAtividades.ShowModal;
end;
```

Ao recompilar a aplicação, será exibida uma mensagem de advertência perguntando se o nome da unit do novo formulário deve ser incluído na cláusula uses da unit UnitPrincipal. Responda "yes" a esta mensagem e recompile o projeto.

Criando um Formulário Para Cadastrar Novas Matrículas

Neste tópico veremos como criar um formulário para acrescentar novas matrículas.

Definindo o Formulário

Este formulário será usado para acrescentar um nova matrícula de um sócio numa atividade à tabela de matrículas. Como já descrito anteriormente, esta tabela possui os seguintes campos:

- CodigoSócio. Este campo, do tipo inteiro, armazena o código de sócio que será matriculado.
- CodigoAtividade. Este campo, do tipo inteiro, armazena o código da atividade na qual o sócio será matriculado.

Estes campos poderiam ser acessados através de um componente do tipo DBEdit. No entanto, o valor armazenado neles deve ser igual a um dos valores já armazenados no campo CodigoSócio (para o campo referente ao sócio que está sendo matriculado) da tabela de Sócios, ou igual a um dos valores já armazenados no campo CodigoAtividade (para o campo referente à atividade na qual o sócio será matriculado) da tabela de Atividades. Desta forma evitamos que se selecione um sócio ou uma atividade não cadastrados. Para resolver este problema, podemos apresentar uma lista de opções usando um componente do tipo DBLookupComboBox, que será apresentado nos próximos tópicos.

Criando o Formulário

Para criar o formulário, execute os seguintes procedimentos:

1. Selecione o item New/Form do menu File do Delphi 7. Será criado um formulário default, chamado Form1.
2. Altere o valor da propriedade Name do formulário para FormCadastraMatriculas.
3. Altere o valor da sua propriedade Caption para "Cadastro de Matrículas".
4. Salve a unit associada a este formulário com o nome UnitCadastraMatriculas.
5. Inclua a unit UnitCadastraMatriculas na cláusula uses da unit UnitPrincipal.

Inserindo os Componentes no Formulário

Para incluir os controles no formulário, execute os seguintes procedimentos:

1. Insira no formulário um componente do tipo Label e defina como "Sócio" o valor da sua propriedade Caption.
2. Insira no formulário um segundo componente do tipo Label e defina como "Atividade" o valor da sua propriedade Caption.
3. Inclua o nome da unit UnitDados na cláusula uses da unit associada a este formulário.
4. Insira no formulário um componente do tipo DBLookupCombobox (da página Data Controls), posicione-o abaixo do componente Label cuja propriedade Caption foi definida como "Sócio" e defina os seguintes valores para as suas principais propriedades:

   Name: DBLookupComboboxSocio.
   DataSource: Dados.DatasouceMatriculas.
   DataField: Socio.
   ListSource: Dados.DatasouceSocios.
   ListField: Nome.
   KeyField: CodigoSocio.

Para definir o sócio que está sendo matriculado, optou-se por usar um componente do tipo DBLookupComboBox, que permite exibir em uma lista drop-down os valores armazenados em um campo dos registros de uma tabela e gravar um valor correspondente ao item selecionado em um campo do registro corrente de uma segunda tabela.

Este componente trabalha, portanto, com duas tabelas: uma tabela-fonte a partir da qual são obtidos os valores a serem exibidos na lista e uma segunda tabela (tabela-destino), na qual será gravado, em um campo do registro corrente, um valor que vai depender do item selecionado na lista.

Para acessar duas tabelas, têm de ser usados, evidentemente, dois componentes do tipo DataSource, que são especificados nas propriedades ListSource (para a tabela-fonte) e DataSource (para a tabela-destino) do componente DBLookupCombobox.

5. Insira no formulário um segundo componente do tipo DBLookupCombobox, posicione-o abaixo do componente Label cuja propriedade Caption foi definida como "Atividade" e defina os seguintes valores para as suas principais propriedades:

Name: DBLookupComboboxAtividade.
DataSource: Dados.DatasouceMatriculas.
DataField: Atividade.
ListSource: Dados.DatasouceAtividades.
ListField: Nome.
KeyField: CodigoAtividade.

Propriedades do Componente DBLookupComboBox

Entre as principais propriedades deste componente, podem-se destacar:

- Name: Define o nome pelo qual o objeto será referenciado no código da aplicação.
- Left e Top: Definem a posição do componente no formulário.
- Height e Width: Definem as dimensões do componente.
- DataSource: Define o componente que estabelece a conexão com a tabela-destino.
- ListSource: Define o componente que estabelece a conexão com a tabela-fonte.
- KeyField: Define o campo de ligação entre as tabelas. Esta propriedade define o campo da tabela-fonte cujo valor do registro corrente será armazenado num campo do registro corrente da tabela-destino (o nome do campo será o especificado na propriedade DataField). O registro corrente da tabela-fonte será o correspondente ao item selecionado na lista.
- DataField: Esta propriedade define o campo do registro corrente da tabela-destino, no qual será gravado o valor do campo especificado na propriedade KeyField do registro corrente da tabela-fonte.
- ListField: Define o campo dos registros da tabela-fonte, cujos valores serão exibidos na lista drop-down do componente.

6. Coloque um botão de comando com figura (BitBtn) para Cadastro, e defina da seguinte maneira as suas principais propriedades:

   Kind: bkCustom.
   ModalResult: mrNone
   Name: BotaoCadastrar
   Caption: &Cadastrar
   Height: 50
   Width: 100

7. Para que estes dados sejam gravados na tabela quando o usuário selecionar este botão, devemos definir da seguinte maneira o procedimento associado ao evento OnClick deste botão.

```
procedure TFormCadastraMatriculas.BotaoCadastrarClick(Sender: TObject);
begin
  try
    Dados.SimpleDatasetMatriculas.Post;
    Dados.SimpleDatasetMatriculas.Append;
  except
    ShowMessage('Este Sócio Já Foi Cadastrado Nesta Atividade');
  end;
  DBLookupComboBoxSocio.SetFocus;
end;
```

Desta maneira, o índice será novamente trocado ao se fechar o formulário.

8. Defina da seguinte maneira o procedimento associado ao evento OnShow deste formulário:

```
procedure TFormCadastraMatriculas.FormShow(Sender: TObject);
begin
    Dados.SimpleDatasetMatriculas.IndexName := 'indMatrcula';
    Dados.SimpleDatasetMatriculas.Open;
    Dados.SimpleDatasetMatriculas.Append;
end;
```

Desta maneira, o índice será trocado ao se exibir o formulário.

9. Defina da seguinte maneira o procedimento associado ao evento OnKeyPress do componente DBLookupComboboxSocio, a ser compartilhado pelos componente DBLookupComboboxSocio.

```
procedure TFormCadastraMatriculas.DBLookupComboBoxSocioKeyPress(
  Sender: TObject; var Key: Char);
begin
     if (key = #13) then
     begin
         Key := #0;
         SelectNext((Sender as TWinControl),True,True);
    end;
end;
```

10. Defina da seguinte maneira o procedimento associado ao evento OnClose deste formulário:

```
procedure TFormCadastraMatriculas.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    Dados.SimpleDatasetMatriculas.Delete;
    Dados.SimpleDatasetMatriculas.ApplyUpdates(0);
    Dados.SimpleDatasetMatriculas.IndexName := 'indSocio';
end;
```

11. Inclua mais um botão de comando com figura e defina suas principais propriedades como indicado a seguir:

    Name: BotaoFechar
    Kind: bkClose
    Caption: &Fechar
    Height: 50
    Width: 100

Seu formulário deverá ficar com o aspecto mostrado na figura a seguir.

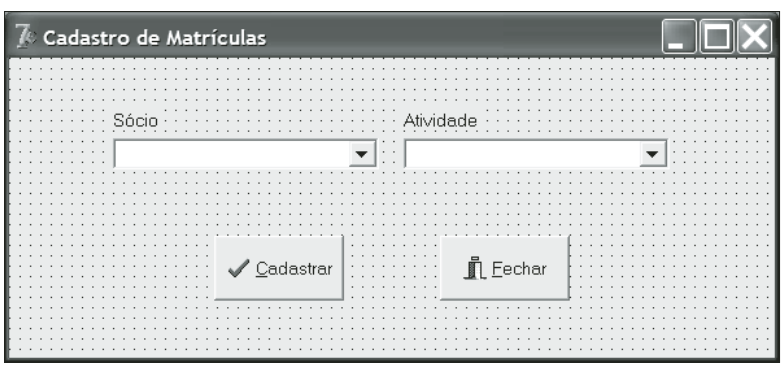


Figura 12.6: Aspecto final do formulário para Cadastro de Atividades.

12. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Cadastrar do menu Matriculas, e compartilhe este procedimento com o evento OnClick do item Cadastrar do submenu Matriculas do menu pop-up:

```
procedure TFormPrincipal.MatriculasCadastroClick(Sender: TObject);
begin
    FormCadastraMatriculas.ShowModal;
end;
```

13. Use a caixa de diálogo Edit Tab Order para que os componentes sejam ordenados na forma mais adequada para a digitação das informações.

Repare que ao exibir o formulário e selecionar a seta à direita do componente DBLookupComboboxSocio, apenas o nome de um sócio é exibido. Isto ocorre porque, em nossa tabela, o nome e o sobrenome de um sócio foram armazenados em campos distintos. E como resolver o caso de vários sócios com o mesmo nome? Neste caso, a solução consiste em empregar o recurso de campos calculados, apresentado no tópico a seguir.

Trabalhando com Campos Calculados

Um campo calculado é um tipo especial de campo de uma tabela, cujo valor depende dos valores armazenados em outros campos. Este campo só existe durante a execução do aplicativo, reduzindo desta maneira a quantidade de espaço em disco a ser ocupado pela tabela.

Podemos então criar um campo calculado na tabela de Sócios, chamado NomeCompleto, cujo valor será obtido pela união dos valores armazenados nos campos Nome e Sobrenome. Para criar este campo calculado, execute os seguintes procedimentos:

1. Exiba o objeto Dados (o DataModule).
2. Selecione o componente SimpleDatasetSocios.
3. Redefina como False o valor da propriedade Active deste componente.
4. Pressione o botão direito do mouse, para exibir o menu pop-up do componente.
5. Selecione o item Fields Editor do menu pop-up, para exibir o Editor de campos.
6. Pressione o botão direito do mouse, para exibir o menu pop-up do Fields Editor.
7. Selecione o item New Field do menu pop-up, para exibir a caixa de diálogo New Field, mostrada na figura a seguir.

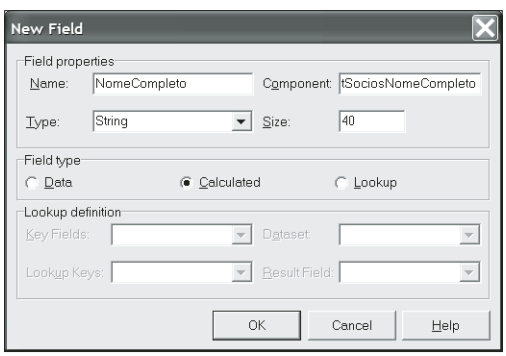


Figura 12.7: Criando um campo calculado.

8. Preencha os campos da caixa de diálogo New Field como indicado na figura anterior.
9. Clique em OK, para fechar a caixa de diálogo e criar o novo campo.
10. Redefina como True o valor da propriedade Active deste componente.
11. Defina da seguinte maneira o procedimento associado ao evento OnCalcFields do componente SimpleDatasetSocios:

```
procedure TDados.SimpleDataSetSociosCalcFields(DataSet: TDataSet);
begin
    SimpleDatasetSociosNomeCompleto.AsString :=
    SimpleDatasetSociosNome.AsString + ' ' +SimpleDatasetSociosSobrenome.AsString;
end;
```

A linha de código anterior faz com que o valor armazenado no campo NomeCompleto seja obtido pela concatenação do valor armazenado no campo Nome, com uma string composta por um espaço em branco e o valor armazenado no campo Sobrenome. Não esqueça de incluir a string composta por um espaço em branco, para que seja inserido um espaço entre o nome e o sobrenome do sócio. Esta string também será útil quando criarmos um formulário para a consulta de dados de sócios.

12. Redefina como NomeCompleto a propriedade ListField do componente DBLookupComboboxSocio.

Capítulo 13

Criando Formulários Para Alteração de Sócios e Atividades

Neste capítulo serão apresentados os procedimentos necessários à criação de formulários para alteração dos dados de Sócios e Atividades.

Serão utilizados muitos dos procedimentos descritos nos capítulos anteriores. Portanto, é importante que estes tenham sido perfeitamente compreendidos pelo leitor.

Criação de Formulários Para Alteração de Dados

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Conhecimentos básicos sobre a linguagem Object Pascal (já apresentados nos capítulos anteriores).
- Criação de formulários e utilização dos componentes básicos de interface.
- Utilização de componentes para acesso a bancos de dados e de visualização.

METODOLOGIA

- Apresentação e descrição dos principais conceitos relacionados à criação de Templates de Componentes para agrupar componentes a serem utilizados em vários Formulários.

TÉCNICA

- Apresentação e descrição dos procedimentos necessários à criação de Templates de Componentes.

O Conceito de Templates de Componentes

Existem situações em que um mesmo grupo de componentes é usado em diversos formulários.

Neste capítulo, por exemplo, criaremos formulários para a alteração dos dados dos sócios e atividades. Para que isso seja possível, devemos, em cada formulário, incluir vários componentes que acessem os diversos campos de cada uma das tabelas. Acontece que este trabalho já foi feito na criação de formulários de cadastro, e seria razoável aproveitar o trabalho já realizado.

A fim de facilitar o trabalho do desenvolvedor, o Delphi apresenta o conceito de Templates de Componentes – entidades capazes de armazenar um conjunto de componentes (e os procedimentos associados a seus eventos) como um tipo especial de componente.

Neste capítulo, criaremos três formulários com uma característica comum: permitir alteração de registros. Conseqüentemente, cada um destes formulários possuirá:

- Os componentes de visualização e labels criados nos respectivos formulários de Cadastro.
- Uma caixa de texto para que o usuário possa pesquisar o registro a ser alterado pelo Nome do Sócio ou Atividade.
- Um botão a ser utilizado para fechar o formulário.
- Um componente DBNavigator, para permitir uma navegação rápida entre os registros.

Criando os Templates de Componentes

Para criar o Template de Componentes de Visualização da tabela de Sócios usando os componentes já existentes no formulário de cadastro de Sócios, você deverá executar os seguintes procedimentos:

1. Exibir o formulário FormCadastraSocios.
2. Selecionar todos os componentes Label, DBEdit e DBComboBox existentes neste formulário.
3. Selecionar o item Create Component Template do menu Component, para exibir a caixa de diálogo mostrada na figura a seguir, na qual deverão ser fornecidas as informações necessárias à criação do template.

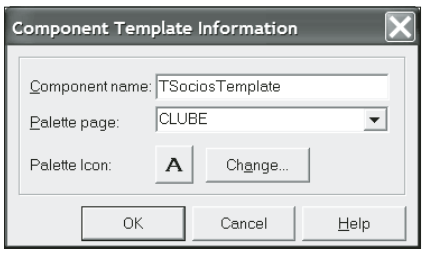


Figura 13.1: A caixa de diálogo Component Template Information, usada para manipular componentes de acesso à tabela de sócios.

4. Nesta caixa de diálogo deverão ser informados:
   - Um nome para o template.
   - O nome da página da paleta de componentes na qual o ícone do Template será exibido como se fosse um componente. Neste exemplo digitaremos o nome de uma página inexistente na paleta de componentes, e que será então automaticamente criada pelo ambiente de desenvolvimento.
   - Um ícone para o Template (neste exemplo, estamos usando o ícone do componente Label).

Digite as informações mostradas na figura anterior e selecione o botão OK, para fechar esta caixa de diálogo e criar o Template de Componentes. Será criada uma página chamada Clube na paleta de componentes, para armazenar o Template recém-criado.

Para criar o Template de Componentes para a tabela Atividades usando os componentes já existentes no formulário de cadastro de Atividades, você deverá executar os seguintes procedimentos:

1. Exibir o formulário FormCadastraAtividades.
2. Selecionar todos os componentes Label e DBEdit existentes neste formulário.
3. Selecionar o item Create component Template do menu Component, para exibir a caixa de diálogo mostrada na figura a seguir, na qual deverão ser fornecidas as informações necessárias à criação do template.

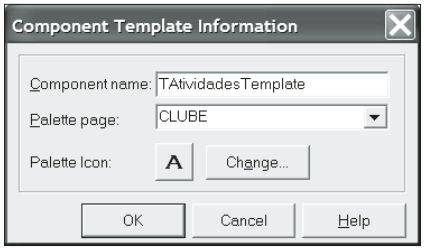


Figura 13.2: A caixa de diálogo Component Template Information, usada para manipular componentes de acesso à tabela de atividades.

4. Digite as informações mostradas na figura anterior e selecione o botão OK, para fechar esta caixa de diálogo e criar o Template de Componentes.

Criando o Formulário de Alteração de Sócios

Para criar o formulário de alteração de sócios, você deve executar os seguintes procedimentos:

1. Selecione o item New/Form do menu File do Delphi 7, para criar um novo formulário.
2. Altere o valor das suas propriedades Name e Caption para FormAlteraSocios e "Alteração de Dados dos Sócios", respectivamente.
3. Salve a unit associada a este formulário com o nome UnitAlteraClientes.
4. Inclua o nome das units UnitDados e DB na cláusula uses da unit associada ao formulário recém-criado.
5. Inclua o Template SociosTemplate no formulário. Repare que todos os componentes serão criados de uma só vez, bem como os procedimentos associados a seus eventos. A inclusão de um Template segue os mesmos procedimentos necessários à inclusão de um componente.
6. Inclua um botão de comando com figura na parte inferior do formulário, e defina da seguinte maneira as suas principais propriedades:

   Name: BotaoFechar
   Kind: bkClose
   Caption: &Fechar
   Height: 50
   Width: 100

7. Inclua um componente Caixa de Texto (Edit) no formulário, e altere o valor da sua propriedade Name para EditPesquisa, apague o valor digitado na sua propriedade Text e codifique da seguinte maneira o procedimento associado ao seu evento OnChange:

```
procedure TFormAteraSocios.EditPesquisaChange(Sender: TObject);
begin
Dados.SimpleDatasetSocios.Locate('NOME',VarArrayOf([EditPesquisa.Text]),[loPartialKey,LocaseInsensitive]);
end;
```

Para evitar erros de compilação mantenha todo o comando digitado em uma única linha.

8. Defina da seguinte maneira o procedimento associado ao evento OnShow do formulário:

```
procedure TFormAteraSocios.FormShow(Sender: TObject);
begin
    Dados.SimpleDatasetSocios.Open;
    EditPesquisa.Clear;
end;
```

9. Defina da seguinte maneira o procedimento associado ao evento OnClose do formulário:

```
procedure TFormAlteraSocios.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    Dados.SimpleDatasetSocios.ApplyUpdates(0);
end;
```

10. Inclua próximo ao componente EditPesquisa um componente Label no formulário e altere o valor da sua propriedade Caption para "DIGITE O NOME A SER PESQUISADO", respectivamente.

11. Inclua um componente DBNavigator (página Data Controls) no formulário e defina da seguinte maneira as suas principais propriedades:

    Name: DBNavigatorSocios
    DataSource: Dados.DatasourceSocios
    VisibleButtons: [nbFirst,nbPrior,nbNext,nbLast]

12. Reposicione os componentes no formulário, de maneira que este fique com um aspecto semelhante ao apresentado na figura a seguir.

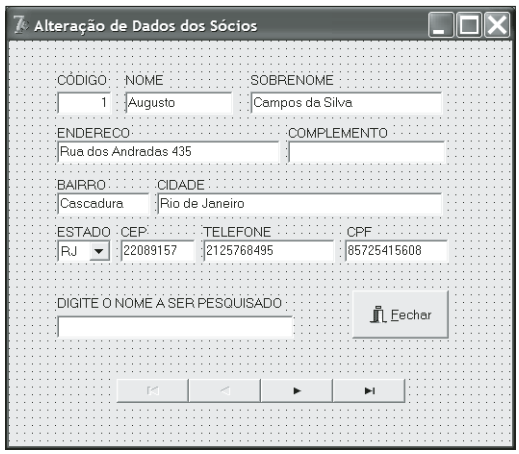


Figura 13.3: Aspecto do formulário para alteração dos dados dos sócios.

13. Inclua o nome da unit associada a este formulário na cláusula uses da unit associada ao formulário FormPrincipal.

14. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item Alteração do menu Sócios, e compartilhe este procedimento com o evento OnClick do item Alteração do submenu Sócios do menu pop-up do formulário.

```
FormAlteraSocios.ShowModal;
```

Criando o Formulário de Alteração de Atividades

Para criar o formulário de alteração de atividades, você deve executar os seguintes procedimentos:

1. Selecione o item New/Form do menu File do Delphi 7, para criar um novo formulário.
2. Altere o valor das suas propriedades Name e Caption para FormAlteraAtividades e "Alteração de Atividades", respectivamente.
3. Salve a unit associada a este formulário com o nome UnitAlteraAtividades.
4. Inclua os nomes das units UnitDados e DB na cláusula uses da unit associada ao formulário recém-criado.
5. Inclua o Template AtividadesTemplate no formulário. Repare que todos os componentes serão criados de uma só vez, bem como os procedimentos associados a seus eventos.
6. Inclua outro botão de comando com figura na parte inferior do formulário, e defina da seguinte maneira as suas principais propriedades:

   Name: BotaoFechar
   Kind: bkClose
   Caption: &Fechar
   Height: 50
   Width: 100
7. Inclua um componente Caixa de Texto (Edit) no formulário, e altere o valor da sua propriedade Name para EditPesquisa, apague o valor digitado na sua propriedade Text e codifique da seguinte maneira o procedimento associado ao seu evento OnChange:

```
procedure TFormAlteraAtividades.EditPesquisaChange(Sender: TObject);
begin
Dados.SimpleDatasetAtividades.Locate('NOME',VarArrayOf([EditPesquisa.Text]),[loPartialKey,LocaseInsensitive]);
end;
```

8. Defina da seguinte maneira o procedimento associado ao evento OnShow do formulário:

```
procedure TFormAlteraAtividades.FormShow(Sender: TObject);
begin
    Dados.SimpleDatasetAtividades.Open;
    EditPesquisa.Clear;
end;
```

9. Defina da seguinte maneira o procedimento associado ao evento OnClose do formulário:

```
procedure TFormAlteraAtividades.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    Dados.SimpleDatasetSocios.ApplyUpdates(0);
end;
```

10. Inclua próximo ao componente EditPesquisa um componente Label no formulário e altere o valor da sua propriedade Caption para "DIGITE O NOME A SER PESQUISADO".
11. Inclua um componente DBNavigator no formulário e defina da seguinte maneira as suas principais propriedades:

    Name: DBNavigatorAtividades
    DataSource: Dados.DatasourceAtividades
    VisibleButtons: [nbFirst,nbPrior,nbNext,nbLast]

12. Reposicione os componentes no formulário, de maneira que este fique com um aspecto semelhante ao apresentado na figura a seguir.

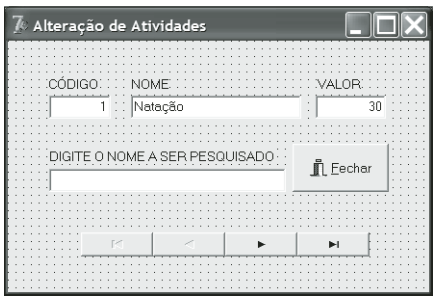


Figura 13.4: Aspecto do formulário para alteração dos Dados das Atividades.

13. Inclua o nome da unit associada a este formulário na cláusula uses da unit associada ao formulário FormPrincipal.
14. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item Alteração do menu Atividades, e compartilhe este procedimento com o evento OnClick do item Alteração do submenu Atividades do menu pop-up do formulário.

```
FormAlteraAtividades.ShowModal;
```

O Componente DBNavigator

Este controle permite a manipulação direta dos registros de uma tabela.

Este controle permite que o usuário manipule os registros de uma tabela do banco de dados e seus botões possuem os seguintes significados (da esquerda para a direita):

- Exibir o primeiro registro da tabela.
- Exibir o registro anterior da tabela.
- Exibir o próximo registro da tabela.
- Exibir o último registro da tabela.
- Insere um registro em branco na tabela.
- Deleta (exclui) o registro corrente da tabela.
- Permite a edição do registro corrente da tabela.
- Grava o registro corrente da tabela.
- Cancela a edição do registro corrente do banco de dados.
- Atualiza a exibição dos registros da tabela.

Entre as propriedades deste componente, podemos destacar:

- DataSource: Esta propriedade define o componente que nos fornecerá a conexão que desejamos estabelecer.

- Name: Esta propriedade define o nome pelo qual o componente será referenciado no programa.
- ShowHint: Essa propriedade é uma variável booleana que define se o componente deve ou não exibir strings de auxílio quando o ponteiro do mouse estiver sobre cada um dos seus botões.
- Hints: Essa propriedade é um objeto da classe TStrings e define as strings de auxílio dos diversos botões exibidos pelo componente DBNavigator (permitindo a sua personalização).

Para personalizar as strings de auxílio dos controles DBNavigator incluídos nos formulários, execute os seguintes procedimentos:

1. Selecione o componente DBNavigator, clicando sobre o mesmo com o botão esquerdo do mouse.
2. Selecione a propriedade Hints diretamente no Object Inspector.
3. Clique com o botão esquerdo do mouse sobre as reticências (...) exibidas do lado direito da propriedade no Object Inspector. Será exibida a caixa de diálogo String List Editor.
4. Digite as expressões mostradas na figura a seguir.

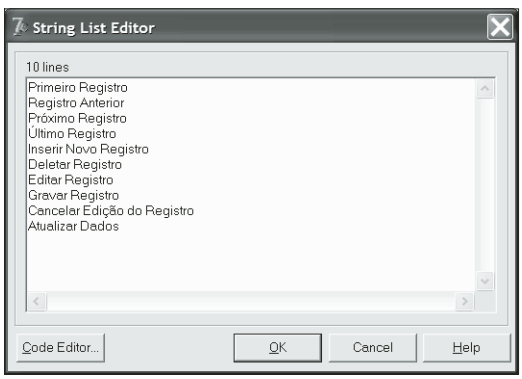


Figura 13.5: Configurando as strings de auxílio para o componente DBNavigator.

5. Selecione o botão OK, para fechar a caixa de diálogo.

Nessa caixa de diálogo, cada string digitada em uma linha corresponde à string de auxílio de um botão. A primeira linha corresponde ao primeiro botão (da esquerda para a direita); a segunda linha, ao segundo botão e assim por diante.

Os métodos correspondentes a cada um desses botões, disponíveis para os objetos da classe TSimpleDataset, são:

- Primeiro Registro: Método First.
- Registro Anterior: Método Prior.
- Próximo Registro: Método Next.
- Último Registro: Método Last.
- Inserir Registro: Método Append.
- Deletar Registro: Método Delete.

- Editar Registro: Método Edit.
- Gravar Registro: Método Post.
- Cancelar Edição do Registro: Método Cancel.
- Atualizar a Exibição do Registro: Método Refresh.
- ShowHint: Esta propriedade é definida como uma variável booleana e define se uma string de auxílio deve ou não ser exibida quando o usuário mantiver o ponteiro do mouse sobre um controle. Se o seu valor for False, nenhuma string de auxílio será exibida durante a execução do aplicativo.
- VisibleButtons: Define os botões que devem ser exibidos no controle.

Você pode alterar essa propriedade mediante a inclusão de uma linha de código ou modificando os valores das subpropriedades de VisibleButtons diretamente no Object Inspector.

Para exibir essas subpropriedades, dê um duplo clique com o botão esquerdo do mouse sobre o sinal (+) exibido à esquerda do nome da propriedade VisibleButtons. As subpropriedades são exibidas (podendo ser diretamente alteradas) e o sinal (+) é substituído por (-). Dando um duplo clique com o botão esquerdo do mouse sobre o símbolo (-), as subpropriedades são novamente ocultadas. Para exibir um botão, atribua o valor True à subpropriedade correspondente, e False em caso contrário.

Lembre-se: a existência de um sinal de (+) imediatamente à esquerda do nome de uma propriedade no Object Inspector indica que essa propriedade tem subpropriedades.

Cada subpropriedade (de cima para baixo) corresponde a um dos botões (da esquerda para a direita).

Neste exemplo, apenas os quatro primeiros botões foram habilitados, usando a propriedade VisibleButtons.

Capítulo 14

Criando Formulários Para Exclusão de Sócios, Atividades e Matrículas

Neste capítulo serão apresentados os procedimentos necessários à criação de formulários para exclusão dos dados de Sócios, Atividades e Matrículas.

Serão utilizados muitos dos procedimentos descritos nos capítulos anteriores. Portanto, é importante que estes tenham sido perfeitamente compreendidos pelo leitor.

Criação de Formulários Para Exclusão de Dados

PRÉ-REQUISITOS

- Noções básicas da utilização dos ambientes de desenvolvimento do Delphi 7.
- Conhecimentos básicos sobre a linguagem Object Pascal (já apresentados nos capítulos anteriores).
- Criação de formulários e utilização dos componentes básicos de interface.
- Utilização de componentes para acesso a bancos de dados e de visualização.

METODOLOGIA

- Apresentação e descrição dos procedimentos necessários à criação de Templates de formulários Templates.
- Apresentação e descrição dos principais conceitos relacionados à criação de Formulários Para Exclusão de Dados.

TÉCNICA

- Descrição dos procedimentos necessários à criação de formulários Para Exclusão.

Criando um Template de Formulário

Neste capítulo, criaremos três formulários com uma característica comum: Excluir um registro selecionado. Conseqüentemente, cada um destes formulários possuirá:

- Um DBGrid conectado a uma tabela através da qual serão visualizados os diversos registros.
- Um botão a ser utilizado para excluir o registro selecionado.
- Uma caixa de texto para que o usuário possa pesquisar o registro a ser excluído pelo Nome do Sócio ou Atividade.
- Um botão a ser utilizado para fechar o formulário.

Para criar um template de formulário para exclusão de registro, você deve executar os seguintes procedimentos:

1. Selecione o item New/Form do menu File do Delphi 7, para criar um novo formulário.
2. Inclua um componente DBGrid (primeiro componente da página Data Controls da paleta de componentes) no formulário.
3. Defina como True a propriedade ReadOnly deste DBGrid, diretamente no Object Inspector.
4. Defina como True a subpropriedade RowSelect da propriedade Options deste componente, diretamente no Object Inspector.
5. Inclua um botão de comando com figura (o segundo componente da página Additional) na parte inferior do formulário, e defina da seguinte maneira as suas principais propriedades:

Name:	BotaoExcluir
Caption:	&Excluir

Glyph: Localize o arquivo Trash.bmp (no Delphi 7 este arquivo costuma estar em c:\Arquivos de programas\Arquivos comuns\Borland shared\Images\Buttons\Trash.bmp, a menos que você tenha alterado o diretório default de instalação).

Height: 50
Width: 100
Default: False
Kind: bkCustom
ModalResult: mrNone

6. Inclua outro botão de comando com figura na parte inferior do formulário, e defina da seguinte maneira as suas principais propriedades:

   Name: BotaoFechar
   Kind: bkClose
   Caption: &Fechar
   Height: 50
   Width: 100

7. Inclua um componente Caixa de Texto (Edit) no formulário, e altere o valor da sua propriedade Name para EditPesquisa.
8. Apague o texto existente na propriedade Text do componente EditPesquisa.
9. Inclua um componente Label no formulário e altere o valor da sua propriedade Caption para "Digite o Nome a ser Pesquisado".
10. Altere as propriedades Name e Caption do formulário para FormExclusao e "Exclusão de ", respectivamente.
11. Reposicione os componentes no formulário, de maneira que este fique com um aspecto semelhante ao apresentado na Figura 14.1.

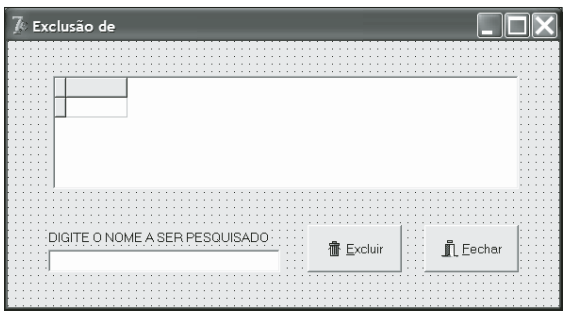


Figura 14.1: Aspecto do formulário a ser usado como template.

12. Inclua o nome da unit UnitDados na cláusula uses da unit associada a este formulário.
13. Selecione o botão direito do mouse sobre o formulário para exibir o seu menu pop-up, conforme mostrado na Figura 14.2.

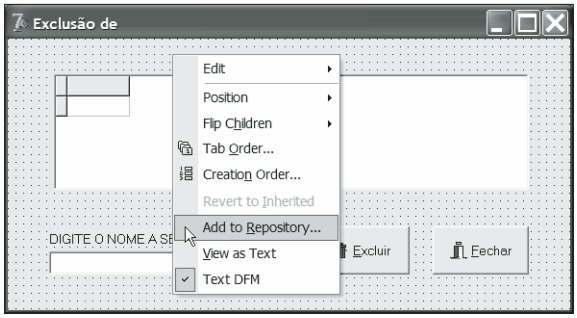


Figura 14.2: Exibindo o menu pop-up do Formulário.

14. Neste menu pop-up, selecione o item Add To Repository, como indicado na figura anterior, para exibir a caixa de diálogo de mesmo nome, reproduzida na Figura 14.3.

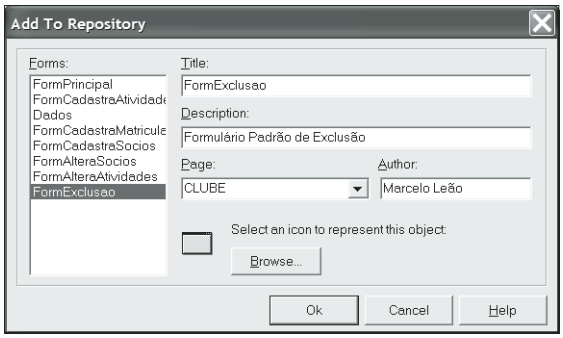


Figura 14.3: A caixa de diálogo Add To Repository.

15. Digite um título para este template na caixa de texto Title, desta caixa de diálogo, como indicado na figura anterior. Este será o nome exibido na caixa de diálogo New Items, quando se desejar criar um novo formulário usando este template.
16. Digite uma descrição para este template na caixa de texto Description, desta caixa de diálogo, como indicado na figura anterior. Esta descrição tem por finalidade documentar o objetivo do template.
17. Na caixa combo Page digite o nome da página da caixa de diálogo New Items na qual o template será exibido, ou selecione um dos nomes disponíveis.
18. Digite o nome do autor deste template na caixa de texto Author, desta caixa de diálogo, como indicado na figura anterior.
19. Opcionalmente, você pode definir um ícone alternativo para representar este template. Para isto, basta selecionar o botão Browse, clicando sobre o mesmo com o botão esquerdo do mouse, e especificar o nome do arquivo do ícone na caixa de diálogo que será exibida.
20. Selecione o botão Ok para fechar esta caixa de diálogo. Será exibida uma mensagem, reproduzida na figura a seguir, indicando que a unit deve ser salva.

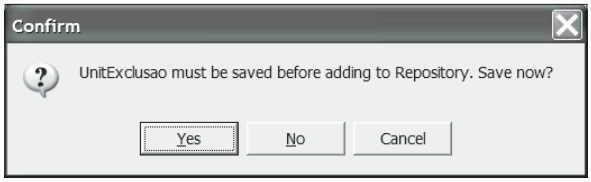


Figura 14.4: Mensagem indicando que a unit deve ser salva.

21. Selecione "yes" nesta caixa de diálogo. Será exibida a caixa de diálogo Save Unit1 As, na qual deverá ser fornecido um nome para a unit. Dê o nome UnitTemplateExclusao para esta unit, como mostrado na figura a seguir e selecione o botão Salvar, para fechar esta caixa de diálogo.

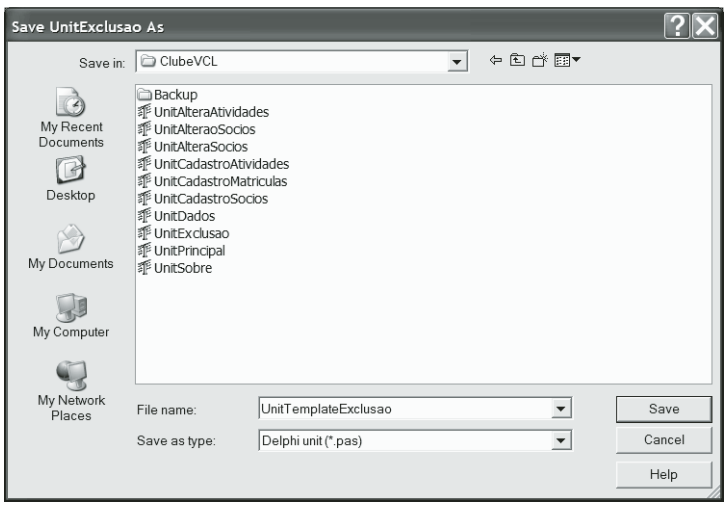


Figura 14.5: A caixa de diálogo Save Unit1 As.

Pronto! O Template foi criado e poderá ser usado na criação dos formulários de exclusão, conforme será mostrado nos próximos tópicos.

Criando Formulários a Partir de um Template

Neste tópico, mostraremos os procedimentos necessários à criação de formulários de exclusão a partir do template criado no tópico anterior.

Criando o Formulário de Exclusão de Sócios

Para criar o formulário de exclusão de sócios a partir do template criado no tópico anterior, você deve executar os seguintes procedimentos:

1. Selecionar o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items. Nesta caixa de diálogo, você deve selecionar o template criado no tópico anterior, como mostrado na Figura 14.4.

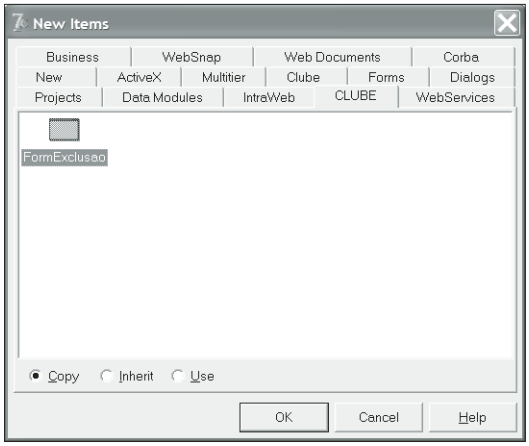


Figura 14.6: A caixa de diálogo New Items.

Repare que esta caixa de diálogo possui a guia Clube (referente ao projeto) e CLUBE (referente ao Template).

2. Selecione o botão OK para fechar esta caixa de diálogo e criar o novo formulário.
3. Altere as propriedades Name e Caption do formulário recém-criado para FormExclusaoSocios e "Exclusão de Sócios", respectivamente.
4. Altere a propriedade Datasource do DBGrid para Dados.DatasourceSocios.
5. Inclua a unit DB na cláusula Uses da unit associada ao formulário.
6. Defina da seguinte maneira o procedimento associado ao evento OnChange do componente EditPesquisa:

```
procedure TFormExclusaoSocios.EditPesquisaChange(Sender: TObject);
begin
Dados.SimpleDatasetSocios.Locate('NOME',VarArrayOf([EditPesquisa.Text]),[loPartialKey,LocaseInsensitive]);
end;
```

Este código faz com que seja efetuada uma pesquisa aproximada pelo nome do cliente.

7. Defina da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoExcluir:

```
procedure TFormExclusaoSocios.BotaoExcluirClick(Sender: TObject);
begin
    if MessageDlg('Confirma a Exclusão do Registro?',mtConfirmation,mbOkCancel,0) = mrOk
    then Dados.SimpleDataSetsocios.Delete;
end;
```

Desta maneira, será enviada uma mensagem de confirmação antes que o registro seja efetivamente excluído.

8. Defina da seguinte maneira o procedimento associado ao evento OnClose deste formulário:

```
procedure TFormExclusaoSocios.FormShow(Sender: TObject);
begin
```

```
    Dados.SimpleDatasetSocios.Open;
end;
```

9. Defina da seguinte maneira o procedimento associado ao evento OnShow deste formulário:

```
procedure TFormExclusaoSocios.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    Dados.SimpleDatasetSocios.ApplyUpdates(0);
end;
```

10. Salve a unit associada a este formulário com o nome UnitExlusaoSocios.
11. Inclua o nome da unit associada a este formulário na cláusula uses da unit associada ao formulário FormPrincipal.
12. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item Exclusão do menu Sócios, e compartilhe este procedimento com o evento OnClick do item Excluir do submenu Sócios do menu pop-up do formulário.

```
FormExclusaoSocios.ShowModal;
```

Criando o Formulário de Exclusão de Atividades

Para criar o formulário de exclusão de atividades a partir do template criado no tópico anterior, você deve executar os seguintes procedimentos:

1. Selecionar o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items. Nesta caixa de diálogo, você deve selecionar o template criado no tópico anterior.
2. Selecione o botão OK para fechar esta caixa de diálogo e criar o novo formulário.
3. Altere as propriedades Name e Caption do formulário recém-criado para FormExclusaoAtividades e "Exclusão de Atividades", respectivamente.
4. Altere a propriedade Datasource do DBGrid para Dados.DatasourceAtividades.
5. Defina da seguinte maneira o procedimento associado ao evento OnShow do formulário:

```
procedure TFormExclusaoAtividades.FormShow(Sender: TObject);
begin
    Dados.SimpleDatasetAtividades.Open;
    EditPesquisa.Clear;
end;
```

6. Inclua a unit DB na cláusula Uses da unit associada ao formulário.
7. Defina da seguinte maneira o procedimento associado ao evento OnChange do componente EditPesquisa:

```
procedure TFormExclusaoAtividades.EditPesquisaChange(Sender: TObject);
begin
Dados.SimpleDatasetAtividades.Locate('NOME',VarArrayOf([EditPesquisa.Text]),[loPartialKey,LocaseInsensitive]);
end;
```

Este código faz com que seja efetuada uma pesquisa aproximada pelo nome da atividade.

8. Defina da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoExcluir:

```
procedure TFormExclusaoAtividades.BotaoExcluirClick(Sender: TObject);
begin
    if MessageDlg('Confirma a Exclusão do Registro?',mtConfirmation,mbOkCancel,0) = mrOk
    then Dados.SimpleDataSetAtividades.Delete;
end;
```

Desta maneira, será enviada uma mensagem de confirmação antes que o registro seja efetivamente excluído.

9. Salve a unit associada a este formulário com o nome UnitExclusaoAtividades.
10. Inclua o nome da unit associada a este formulário na cláusula uses da unit associada ao formulário FormPrincipal.
11. Defina da seguinte maneira o procedimento associado ao evento OnClose deste formulário:

```
procedure TFormExclusaoAtividades.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    Dados.SimpleDatasetAtividades.ApplyUpdates(0);
end;
```

12. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item Exclusão do menu Atividades, e compartilhe este procedimento com o evento OnClick do item Exclusão do submenu Atividades do menu pop-up do formulário.

```
FormExclusaoAtividades.ShowModal;
```

A Linguagem SQL

SQL é a abreviatura de "Structured Query Language" (Linguagem Estruturada de Consulta). Nos últimos anos, a linguagem SQL se transformou numa linguagem padrão para a consulta e acesso a registros de bancos de dados, sendo convencionalmente subdividida em:

- Uma linguagem de manipulação de dados (DML – Data Manipulation Language).
- Uma linguagem de controle de dados (DCL – Data Control Language).
- Uma linguagem de definição de dados (DDL – Data Definition Language).

Embora existam vários dialetos para a linguagem SQL, uma tentativa de padronização foi feita com a definição da linguagem SQL ANSI-92. Este livro não tem por objetivo apresentar ao leitor uma descrição completa da linguagem SQL, mas isto não nos impede de mostrar como os seus comandos podem ser executados a partir de um aplicativo desenvolvido com o Delphi 7.

Para executar comandos SQL em um aplicativo desenvolvido com o Delphi 7, você pode usar o componente SimpleDataset com o qual já está trabalhando, mas deve no entanto considerar as seguintes modificações:

- Sua propriedade CommandTupe deverá ser definida como ctQuery, e não como ctTable.
- Sua propriedade CommandText deverá definir o comando SQL a ser executado.

Criando o Formulário de Exclusão de Matrículas

Para criar o formulário de exclusão de matrículas a partir do template criado no tópico anterior, seguiremos um procedimento um pouco diferenciado. Como a tabela de matrículas só armazena os códigos do sócio e das atividades, e seria difícil pesquisar um sócio ou uma atividade por um destes campos, faremos o acesso através de um comando SQL (uma linguagem padrão para acesso a bancos de dados). Será apresentado também o conceito de relacionamento entre tabelas na memória.

Desta maneira, para criar o novo formulário você deverá executar os seguintes procedimentos:

1. Selecionar o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items. Nesta caixa de diálogo, você deve selecionar o template criado no tópico anterior.
2. Selecione o botão OK para fechar esta caixa de diálogo e criar o novo formulário.
3. Altere as propriedades Name e Caption do formulário recém-criado para FormExclusaoMatriculas e "Exclusão de Matrículas", respectivamente.
4. Inclua um componente SimpleDataset no formulário (e não no DataModule, pois este componente não será compartilhado entre vários formulários) e atribua os seguintes valores às suas principais propriedades:

   Name: SimpleDatasetConsulta.
   Subpropriedade CommandType da Propriedade Dataset: ctQuery
   Subpropriedade Command da Propriedade Dataset: Defina esta propriedade como mostrado na caixa de diálogo Command Text Editor mostrada na Figura a seguir, exibida quando se selecionam os três pontinhos à direita do nome da propriedade.

Subpropriedade Active da Propriedade Dataset: True.

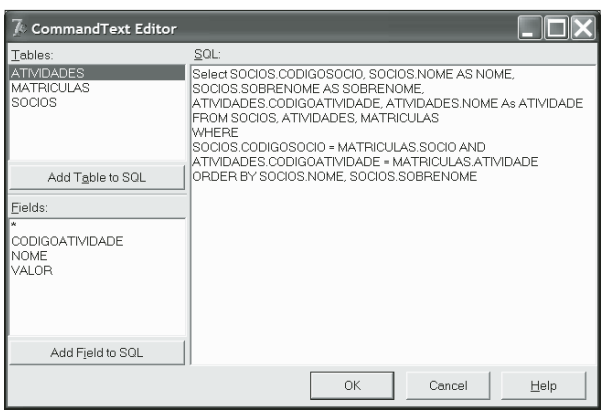


Figura 14.7: A caixa de diálogo Command Text Editor.

- Repare que esta caixa de diálogo define o seguinte comando SQL:

```
Select SOCIOS.CODIGOSOCIO, SOCIOS.NOME AS NOME, SOCIOS.SOBRENOME AS SOBRENOME,
ATIVIDADES.CODIGOATIVIDADE, ATIVIDADES.NOME As ATIVIDADE FROM SOCIOS, ATIVIDADES,
MATRICULAS
WHERE
SOCIOS.CODIGOSOCIO = MATRICULAS.SOCIO AND ATIVIDADES.CODIGOATIVIDADE =
```

```
MATRICULAS.ATIVIDADE
ORDER BY SOCIOS.NOME, SOCIOS.SOBRENOME
```

Vamos analisar este código:

```
Select SOCIOS.CODIGOSOCIO, SOCIOS.NOME AS NOME, SOCIOS.SOBRENOME AS SOBRENOME,
ATIVIDADES.CODIGOATIVIDADE, ATIVIDADES.NOME As ATIVIDADE FROM SOCIOS, ATIVIDADES, MATRICULAS
```

Este trecho de código seleciona (SELECT) os campos CodigoSocio da tabela Sócios (SOCIOS.CODIGOSOCIO), Nome da tabela Sócios (SOCIOS.NOME AS NOME), Sobrenome da tabela Sócios (SOCIOS.SOBRENOME AS SOBRENOME), CodigoAtividade da tabela Atividades (ATIVIDADES.CODIGOATIVIDADE), e campo Nome da tabela Atividades (ATIVIDADES.NOME AS ATIVIDADE).

Repare que, quando se manipulam várias tabelas em um código Select, o acesso a um campo de uma tabela segue a sintaxe Nome_Tabela.Nome_Campo.

O conector As dá um nome alternativo para cada um dos campos selecionados.

```
FROM SOCIOS, ATIVIDADES, MATRICULAS
```

Este trecho de código informa os nomes das tabelas a partir das quais será montada a consulta.

```
WHERE
SOCIOS.CODIGOSOCIO = MATRICULAS.SOCIO AND ATIVIDADES.CODIGOATIVIDADE = MATRICULAS.ATIVIDADE
```

Este trecho de código define as condições que devem ser obedecidas por estes campos. Os campos selecionados são descritos no trecho de código anterior (SELECT) e desde que (WHERE) as seguintes condições sejam satisfeitas:

- O valor armazenado no campo CodigoSocio da tabela Sócios deve ser igual ao campo SOCIO da tabela MATRICULAS.

  ```
  E (AND)
  ```

- O valor armazenado no campo CodigoAtividade da tabela Atividades deve ser igual ao campo ATIVIDADE da tabela MATRICULAS.

  ```
  ORDER BY SOCIOS.NOME, SOCIOS.SOBRENOME
  ```

Este trecho de código define que os registros retornados (satisfazendo as condições anteriores) deverão ser ordenados pelos campos Nome e Sobrenome da tabela Sócios.

5. Inclua um componente Datasource neste formulário e atribua os seguintes valores para as suas principais propriedades:

   Name: DatasourceConsulta.
   Dataset: SimpleDatasetConsulta.

6. Inclua um segundo componente SimpleDataset neste formulário e atribua os seguintes valores às suas principais propriedades:

   Name: SimpleDatasetExclusao.
   Subpropriedade CommandType da Propriedade Dataset: ctTable.
   Subpropriedade Command da Propriedade Dataset: Matriculas.
   Propriedade Mastersource: DatasourceConsulta.

Propriedade Masterfield: Definida selecionando os três pontinhos à direita da propriedade para exibir a caixa de diálogo Field Link Editor, como mostrado nas figuras a seguir. Repare que você deve selecionar os campos que estabelecerão o relacionamento e o botão Add.
Subpropriedade Active da Propriedade Dataset: True.

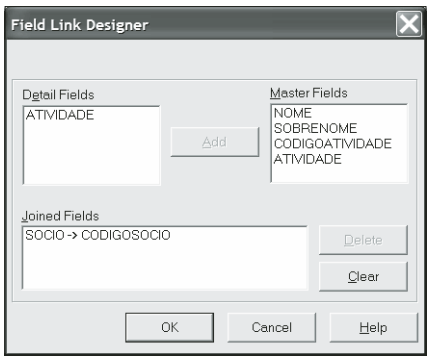


Figura 14.8: Selecionando o primeiro relacionamento.

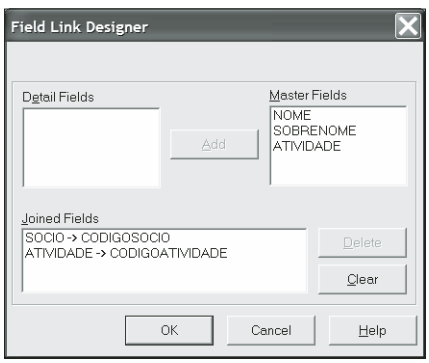


Figura 14.9: Selecionando o segundo relacionamento.

7. Altere a propriedade Datasource do DBGrid para DatasourceConsulta.
8. Defina da seguinte maneira o procedimento associado ao evento OnShow do formulário:

```
procedure TFormExclusaoMatriculas.FormShow(Sender: TObject);
begin
    SimpleDataSetConsulta.Open;
    SimpleDataSetExclusao.Open;
    EditPesquisa.Clear;
end;
```

9. Inclua a unit DB na cláusula Uses da unit associada ao formulário.
10. Defina da seguinte maneira o procedimento associado ao evento OnChange do componente EditPesquisa:

```
procedure TFormExclusaoMatriculas.EditPesquisaChange(Sender: TObject);
begin
Dados.SimpleDatasetConsulta.Locate('Nome',VarArrayOf([EditPesquisa.Text]),[loPartialKey,LocaseInsensitive]);
end;
```

Este código faz com que seja efetuada uma pesquisa aproximada pelo nome do sócio.

11. Defina da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoExcluir:

```
procedure TFormExclusaoMatriculas.BotaoExcluirClick(Sender: TObject);
begin
    if MessageDlg('Confirma a Exclusão do Registro?',mtConfirmation,mbOkCancel,0) = mrOk
    then
    begin
      SimpleDataSetExclusao.Delete;
      SimpleDataSetConsulta.Refresh;
    end;
end;
```

Desta maneira, será enviada uma mensagem de confirmação antes que o registro seja efetivamente excluído.

12. Salve a unit associada a este formulário com o nome UnitExlusaoMatriculas.
13. Inclua o nome da unit associada a este formulário na cláusula uses da unit associada ao formulário FormPrincipal.
14. Defina da seguinte maneira o procedimento associado ao evento OnClose deste formulário:

```
procedure TFormExclusaoMatriculas.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
    Dados.SimpleDatasetExclusao.ApplyUpdates(0);
end;
```

15. Dê um duplo clique com o botão esquerdo do mouse sobre o DBGrid, para exibir a janela Columns Editor (editor de colunas) do DBGrid, como mostrado na figura a seguir.

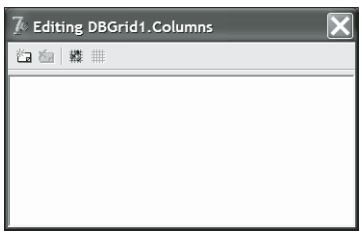


Figura 14.10: O editor de colunas do Componente DBGrid.

16. Exiba o menu pop-up desta janela e selecione o item Add All Fields, para criar objetos que representam os campos exibidos nas colunas do DBGrid, como mostrado na Figura 14.11. Estes objetos permitem que se configure de forma independente cada uma das colunas do DBGrid.
17. Remova os objetos correspondentes aos campos CODIGOSOCIO e CODIGOATIVIDADE, selecionando-se e pressionando a tecla Del. Desta forma, o DBGrid só exibirá as colunas correspondentes aos campos Lookup.

18. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item Exclusão do menu Matrículas, e compartilhe este procedimento com o evento OnClick do item Exclusão do submenu Matrículas do menu pop-up do formulário.

```
FormExclusaoMatriculas.ShowModal;
```

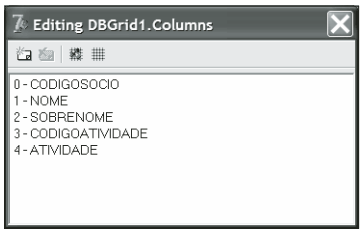


Figura 14.11: O editor de colunas do componente DBGrid.

Capítulo 15

Criando Formulários Para Consulta de Sócios, Atividades e Matrículas

Neste capítulo vamos criar formulários destinados à consulta dos dados dos sócios, atividades e matrículas. Estes formulários serão muito semelhantes àqueles criados para exclusão, exceto pela inexistência de um botão de exclusão. Podemos então criá-los a partir dos formulários de consulta, excluindo o botão "Excluir" em cada um deles.

Criação de Formulários Para Consulta de Dados

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Conhecimentos básicos sobre a linguagem Object Pascal (já apresentados nos capítulos anteriores).
- Criação de formulários e utilização dos componentes básicos de interface.
- Utilização de componentes para acesso a bancos de dados e de visualização.

METODOLOGIA

- Apresentação e descrição dos conceitos relacionados à criação de formulários para consulta de dados.

TÉCNICA

- Apresentação e descrição dos procedimenttos relacionados à criação de formulários para consulta de dados a partir de formulários criados para exclusão.

Criando um Formulário Para a Consulta de Dados dos Sócios

Este formulário será criado a partir do formulário criado para exclusão de sócios, e para isso você deve executar os seguintes procedimentos:

1. Selecionar o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items. Nesta caixa de diálogo, você deve selecionar o Formulário de exclusão de sócios já existente, como mostrado na figura a seguir.

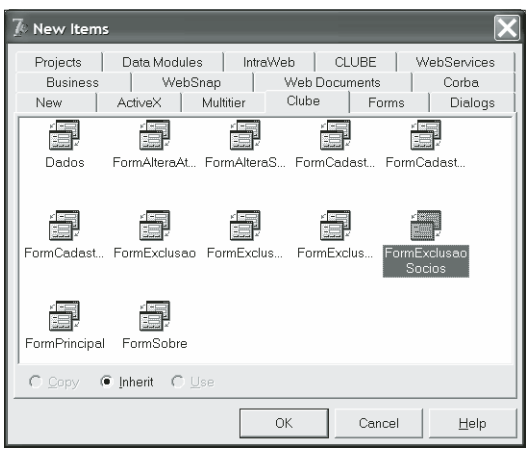


Figura 15.1: A caixa de diálogo New Items.

Repare que neste caso estamos usando a guia Clube (referente ao projeto) e não a guia CLUBE (referente ao Template). Repare ainda a seleção da opção inherit, que indica que será usado o mecanismo de herança de formulários.

2. Selecione o botão OK para fechar esta caixa de diálogo e criar o novo formulário.
3. Altere as propriedades Name e Caption do formulário recém-criado para FormConsultaSocios e "Consulta de Sócios", respectivamente.
4. Defina como False a Propriedade Visible do componente BotaoExcluir.

Repare que, como este formulário foi criado a partir do formulário FormExclusaoSocios usando o mecanismo de herança, não é possível excluir o componente BotaoExcluir, a menos que isso fosse feito no próprio formulário ancestral (não é o caso). A redefinição de valores de propriedade, no entanto, é permitida. A existência deste mecanismo é verificada na seguinte linha de código, extraída da unit deste formulário:

```
type
  TFormConsultaSocios = class(TFormExclusaoSocios)
```

Repare que os diversos componentes estão todos definidos no formulário ancestral. No caso do capítulo anterior, em que foi criado um template independente, a opção Copy pode ser usada na caixa de diálogo New Items, e neste caso foram criadas cópias do formulário criado como template, desvinculados de qualquer herança com o formulário original.

5. Salve a unit associada a este formulário com o nome UnitCONSULTASocios.
6. Inclua o nome da unit associada a este formulário na cláusula uses da unit associada ao formulário FormPrincipal.
7. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item Consulta do menu Sócios, e compartilhe este procedimento com o evento OnClick do item Consulta do submenu Sócios do menu pop-up do formulário.

```
FormConsultaSocios.ShowModal;
```

Criando um formulário Para a Consulta de Dados das Atividades

Este formulário será criado a partir do formulário criado para exclusão de atividades, e para isso você deve executar os seguintes procedimentos:

1. Selecionar o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items. Nesta caixa de diálogo, você deve selecionar o Formulário de exclusão de atividades já existente
2. Selecione o botão OK para fechar esta caixa de diálogo e criar o novo formulário.
3. Altere as propriedades Name e Caption do formulário recém-criado para FormConsultaAtividades e "Consulta de Atividades", respectivamente.
4. Defina como False a Propriedade Visible do componente BotaoExcluir.
5. Salve a unit associada a este formulário com o nome UnitConsultaAtividades.

6. Inclua o nome da unit associada a este formulário na cláusula uses da unit associada ao formulário FormPrincipal.
7. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item Consulta do menu Atividades, e compartilhe este procedimento com o evento OnClick do item Consulta do submenu Atividades do menu pop-up do formulário.

```
FormConsultaAtividades.ShowModal;
```

Criando um Formulário Para a Consulta de Dados das Matrículas

Este formulário será criado a partir do formulário criado para exclusão de Matrículas, e para isso você deve executar os seguintes procedimentos:

1. Selecionar o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items. Nesta caixa de diálogo, você deve selecionar o Formulário de exclusão de Matrículas já existente
2. Selecione o botão OK para fechar esta caixa de diálogo e criar o novo formulário.
3. Altere as propriedades Name e Caption do formulário recém-criado para FormConsultaMatriculas e "Consulta de Matrículas", respectivamente.
4. Defina como False a Propriedade Visible do componente BotaoExclui.
5. Salve a unit associada a este formulário com o nome UnitConsultaMatriculas.
6. Inclua o nome da unit associada a este formulário na cláusula uses da unit associada ao formulário FormPrincipal.
7. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item Consulta do menu Matrículas, e compartilhe este procedimento com o evento OnClick do item Consulta do submenu Matrículas do menu pop-up do formulário.

```
FormConsultaMatriculas.ShowModal;
```

Capítulo 16

Criando Rotinas de Backup e Restauração

Neste capítulo vamos criar as rotinas de backup e restauração de nossos arquivos de dados. Este procedimento é muito importante, tendo em vista que não é raro acontecerem situações em que ocorrem perdas de dados devido a fatores indesejáveis como corte no fornecimento de energia elétrica ou falha no sistema operacional, dentre outras causas possíveis. Evidentemente os procedimentos são muito limitados, devendo-se usar técnicas de compressão de arquivos em aplicações mais sofisticadas.

Cópia de Arquivos

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Delphi 7.
- Conhecimentos básicos sobre a linguagem Object Pascal (já apresentados nos capítulos anteriores).
- Criação de formulários e utilização dos componentes básicos de interface.

METODOLOGIA

- Apresentação e descrição dos comandos da linguagem pascal que permitem a realização de cópias de arquivos.

TÉCNICA

- Apresentação e descrição dos procedimentos necessários à criação de formulários para cópia e restauração de arquivos.

Conceitos Fundamentais

Para efetuar cópias de arquivos, o Delphi 7 fornece acesso a uma função chamada CopyFileTo, definida na unit IdGlobal, que permite que esta tarefa seja executada de maneira simples e rápida.

Esta função recebe como parâmetros:

- Uma string que define o nome do arquivo a ser copiado (incluindo seu path).
- Uma string que define o nome do arquivo de destino – para o qual a cópia será feita (incluindo seu path).

Esta função é multiplataforma, e retorna False se o arquivo de destino já existir. Logo, deve-se inicialmente testar se o referido arquivo já existe e, em caso positivo, apagá-lo.

Criando um Formulário de Backup

Para criar um formulário de Backup, você deverá executar os seguintes procedimentos:

1. Selecione o item New/Form do menu File do Delphi 7, para criar um novo formulário.
2. Altere os valores das suas propriedades Name e Caption para FormBackup e "Backup de Dados" respectivamente.
3. Coloque um label no formulário e defina sua propriedade Caption como "Digite o Nome do Arquivo de Destino (incluindo seu path)".
4. Inclua um componente Caixa de Texto (página Standard) no formulário e defina os seguintes valores para as suas principais propriedades.

Name:	EditDestino
Text	(Deixe em Branco)

5. Inclua um componente Botão de Comando com figura no formulário e defina os seguintes valores para as suas principais propriedades.

   Name: BotaoCopiar
   Caption: &Copiar
   Glyph: c:\Arquivos de programas\Arquivos comuns\Borland shared\Images\Buttons\Floppy.bmp.
   Height: 50
   Width: 100

6. Inclua um segundo componente Botão de Comando com figura no formulário e defina os seguintes valores para as suas principais propriedades.

   Name: BotaoFechar
   Kind: bkClose
   Caption: &Fechar
   Height: 50
   Width: 100

Seu formulário deverá ficar com o aspecto indicado na figura a seguir.

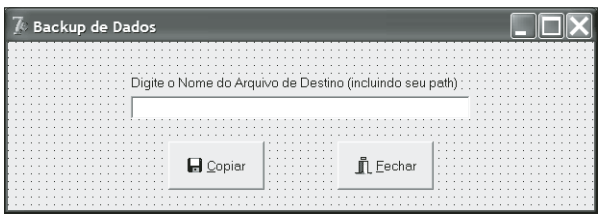


Figura 16.1: Aspecto final do formulário de backup.

7. Inclua o nome da unit IdGlobal na cláusula uses da unit associada a este formulário.
8. Defina da seguinte maneira o procedimento associado ao evento Onclick do componente BotaoCopiar:

```
procedure TFormBackup.BotaoCopiarClick(Sender: TObject);
var
  origem, destino: string;
begin
  origem := ExtractFilePath(Application.Exename)+'C:\DB\Clube.gdb';;
  destino := EditDestino.Text;
  if FileExists(destino) then DeleteFile(destino);
  if (CopyFileTo(origem,destino))
  then
     ShowMessage('Backup Efetuado com Sucesso')
  else
     ShowMessage('Cópia Não Efetuada');
end;
```

A função ExtractFilePath retorna na forma de uma string o diretório completo do arquivo cujo nome é passado como parâmetro, também na forma de uma string. Para obter o nome do arquivo executável da aplicação, basta usar o método ExeName do objeto Application (que representa a aplicação).

Observe que inicialmente são declaradas duas variáveis do tipo string, destinadas a armazenar os nomes dos arquivos de origem e de destino, a serem usadas na chamada da função CopyFileTo.

9. Salve a unit associada a este arquivo de código com o nome UnitBackup.
10. Inclua o nome desta unit na cláusula uses da unit associada ao formulário FormPrincipal (o formulário principal da aplicação).
11. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item Backup do menu Sistema, e compartilhe este procedimento com o evento OnClick do item Backup do submenu Sistema do menu pop-up do formulário.

```
FormBackup.ShowModal;
```

Para criar um formulário de Restauração de dados, você deverá executar os seguintes procedimentos:

1. Selecione o item New/Form do menu File do Delphi 7, para criar um novo formulário.
2. Altere os valores das suas propriedades Name e Caption para FormRestaura e "Recuperação de Dados" respectivamente.
3. Coloque um label no formulário e defina sua propriedade Caption como "Digite o Nome do Arquivo de Origem (incluindo seu path)".
4. Inclua um componente Caixa de Texto (página Standard) no formulário e defina os seguintes valores para as suas principais propriedades.

Name:	EditOrigem
Text	(Deixe em Branco)

5. Inclua um componente Botão de Comando com figura no formulário e defina os seguintes valores para as suas principais propriedades.

   Name: BotaoCopiar
   Caption: &Copiar
   Glyph: c:\Arquivos de programas\Arquivos comuns\Borland shared\Images\Buttons\Floppy.bmp.
   Height: 50
   Width: 100

6. Inclua um segundo componente Botão de Comando com figura no formulário e defina os seguintes valores para as suas principais propriedades.

   Name: BotaoFechar
   Kind: bkClose

Caption: &Fechar
Height: 50
Width: 100

Seu formulário deverá ficar com o aspecto indicado na Figura 16.2.

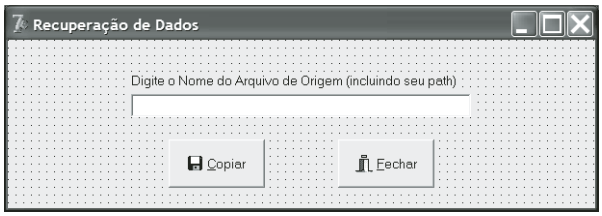


Figura 16.2: Aspecto final do formulário de Recuperação de Dados.

7. Inclua o nome da unit IdGlobal na cláusula uses da unit associada a este formulário.
8. Defina da seguinte maneira o procedimento associado ao evento Onclick do componente BotaoCopiar:

```
procedure TFormRestaura.BotaoCopiarClick(Sender: TObject);
var
  origem, destino: string;
begin
  origem := Editorigem.Text;
  destino := ExtractFilePath(Application.Exename)+'C:\DB\Clube.gdb';;
  if FileExists(destino) then DeleteFile(destino);
  if (CopyFileTo(origem,destino))
  then
     ShowMessage('Recuperação Efetuada com Sucesso')
  else
     ShowMessage('Cópia Não Efetuada');
end;
```

9. Salve a unit associada a este arquivo de código com o nome UnitRestaura.
10. Inclua o nome desta unit na cláusula uses da unit associada ao formulário FormPrincipal (o formulário principal da aplicação).
11. Inclua a seguinte linha de código no procedimento associado ao evento OnClick do item Restaurar do menu Sistema, e compartilhe este procedimento com o evento OnClick do item Restaurar do submenu Sistema do menu pop-up do formulário.

```
FormRestaura.ShowModal;
```

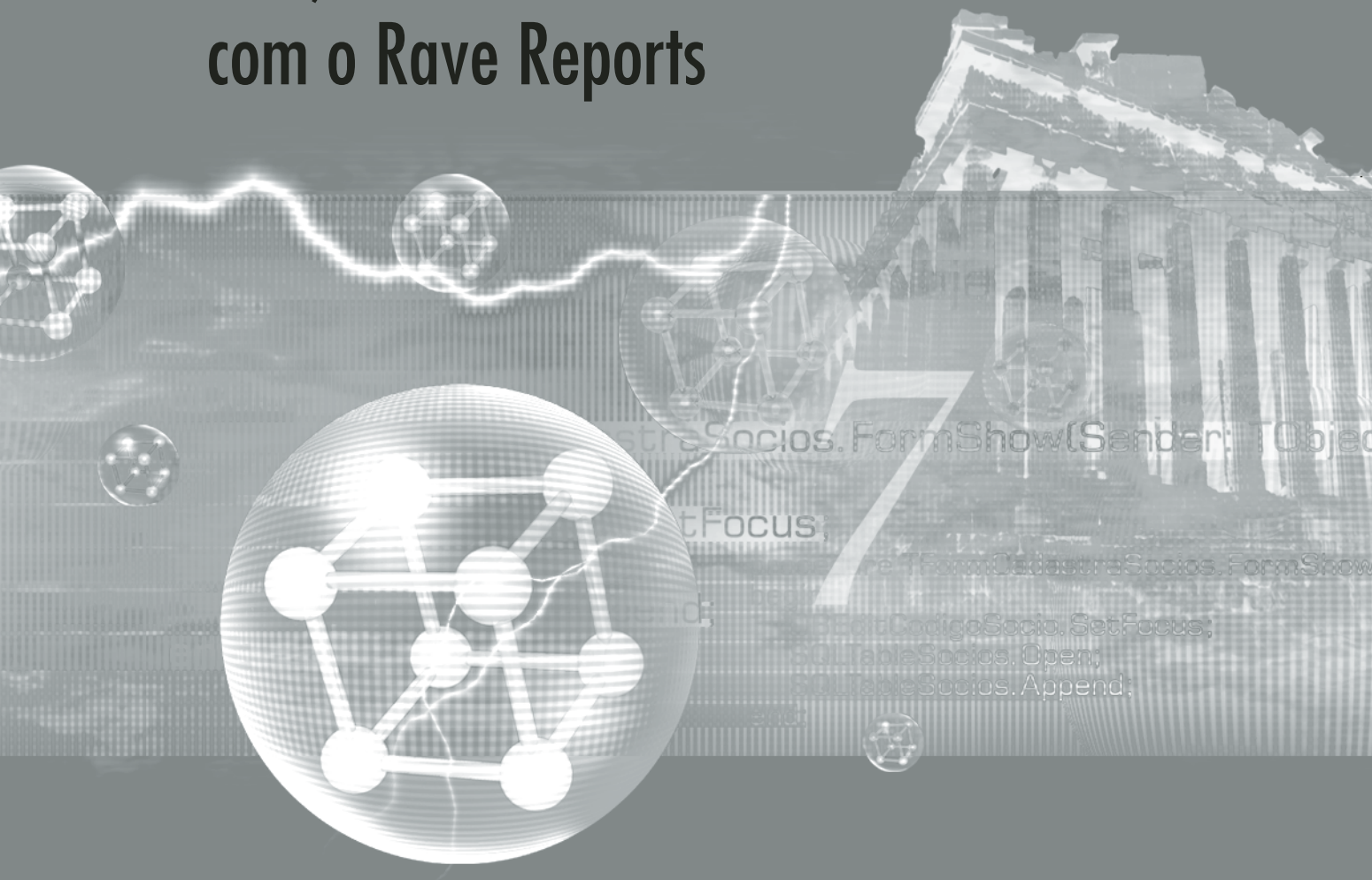
Evidentemente estamos supondo que o backup esteja sendo feito para um meio de armazenamento secundário de grande capacidade, como um ZipDisk ou um SuperDisk. Uma opção alternativa seria a utilização de componentes disponíveis no mercado, que gerenciam a compactação de arquivos (no formato .zip) e possuem métodos capazes de gerenciar cópia de arquivos para múltiplos disquetes.

Parte II

Know-How

Capítulo 17

Criação de Relatórios com o Rave Reports



Neste capítulo, serão apresentados os componentes que integram o pacote do Rave Reports (utilizado para a criação de relatórios em aplicações desenvolvidas com o Borland Delphi 7), e os procedimentos necessários à criação de relatórios com esses componentes.

Know-How em: Criação de Relatórios com o Rave Reports

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Borland Delphi 7.
- Conceitos básicos sobre a utilização de componentes de acesso a bancos de dados.

METODOLOGIA

- Apresentação e descrição dos principais conceitos relacionados à criação de relatórios com o conjunto de componentes Rave Reports.

TÉCNICA

- Apresentação dos procedimentos necessários à criação de relatórios com o conjunto de componentes Rave Reports.

Introdução

O Rave Reports é, na realidade, um conjunto de componentes situados na página Rave da paleta de componentes, que permite a criação de relatórios de forma simples e rápida.

O Componente RvProject

O componente RvProject (o primeiro componente da página Rave da paleta de componentes) é de fundamental importância na criação de relatórios com o Rave Reports, pois fornece acesso direto a um ambiente de desenvolvimento intrínseco à criação de relatórios.

Criando um Relatório de Sócios

Para criar um relatório com os dados dos sócios, você deve executar os seguintes procedimentos:

1. Crie um novo Datamodule, executando os procedimentos já descritos nos capítulos anteriores, e altere sua propriedade Name para Relatórios (lembre-se que um Datamodule é um repositório de objetos não-visuais, e não serve apenas para armazenar componentes de acesso a dados).
2. Inclua um componente RvProject neste Datamodule, e altere o valor da sua propriedade Name para RvClube.
3. Inclua um componente RvDatasetConnection neste Datamodule, e altere o valor da sua propriedade Name para RvDatasetConnectionSocios.
4. Inclua a unit UnitDados na cláusula uses da unit associada a este Datamodule.
5. Atribua o valor Dados.SimpleDatasetSocios à propriedade Dataset do componente RvDatasetConnectionSocios. Seu Datamodule deverá ficar com o aspecto mostrado na figura a seguir.

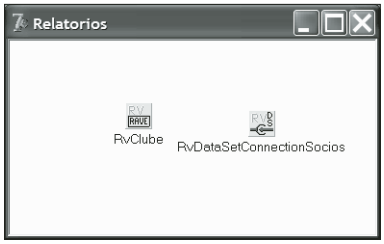


Figura 17.1: Iniciando a criação de um relatório.

6. Dê um duplo clique com o botão esquerdo do mouse sobre o componente RvClube. Será exibido o ambiente de desenvolvimento do Rave Reports, mostrado na figura a seguir.

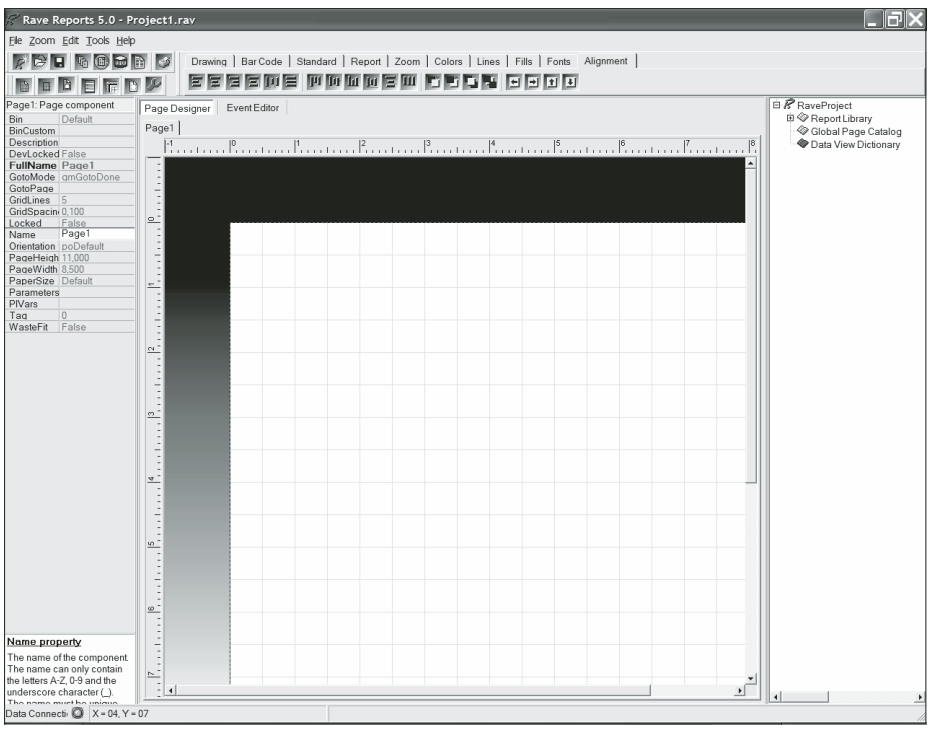


Figura 17.2: O ambiente de desenvolvimento do Rave Reports.

Repare que este ambiente apresenta sua própria paleta de componentes, seu próprio "Object Inspector", etc.

7. Selecione o item New Data Object do menu File do ambiente do Rave Reports, para exibir a caixa de diálogo Data Connection, mostrada na Figura 17.3.

8. Nesta caixa de diálogo, selecione o item Direct Data View e o botão Next. Serão exibidas as conexões de dados disponíveis (no momennto só há uma), como mostra a Figura 17.4.

9. Selecione a única conexão disponível e o botão Finish. Selecione e expanda a opção Data View Dictionary, no lado direito do ambiente do Rave Reports. Selecione o item Dataview1 e altere para DataviewSocios a sua propriedade Name. Repare que foram inseridos objetos correspondentes aos diversos campos da tabela Sócios, como mostrado na figura a seguir.

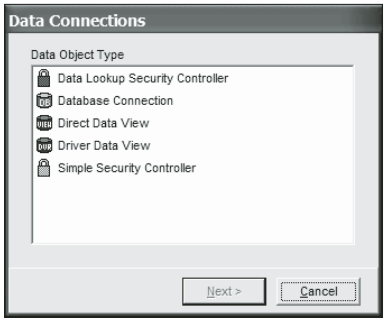


Figura 17.3: A caixa de diálogo Data Connections.

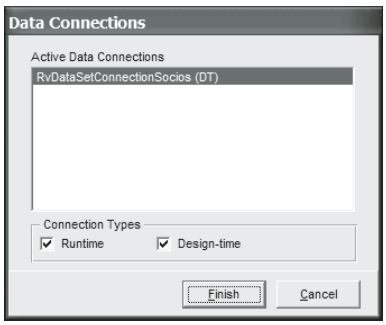


Figura 17.4: Exibindo as conexões de dados disponíveis na caixa de diálogo Data Connections.

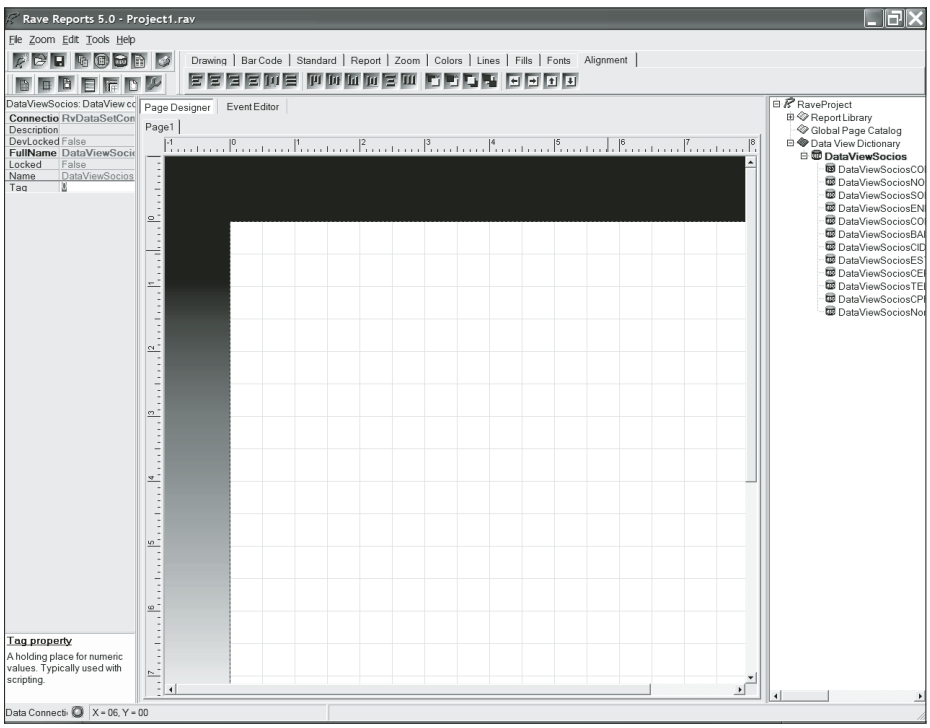


Figura 17.5: Exibindo os objetos correspondentes à opção Data View Dictionary.

10. Selecione e expanda a opção Repot Library, no lado direito do ambiente do Rave Reports. Selecione o item Report1 e altere para RelatSocios a sua propriedade Name.

11. Inclua um componente Region (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) na área da página do relatório (área central do ambiente de desenvolvimento do Rave Reports) e faça com que este componente ocupe toda a área disponível, como mostrado na figura a seguir. Este componente permite que sejam adicionadas bandas (faixas) ao relatório.

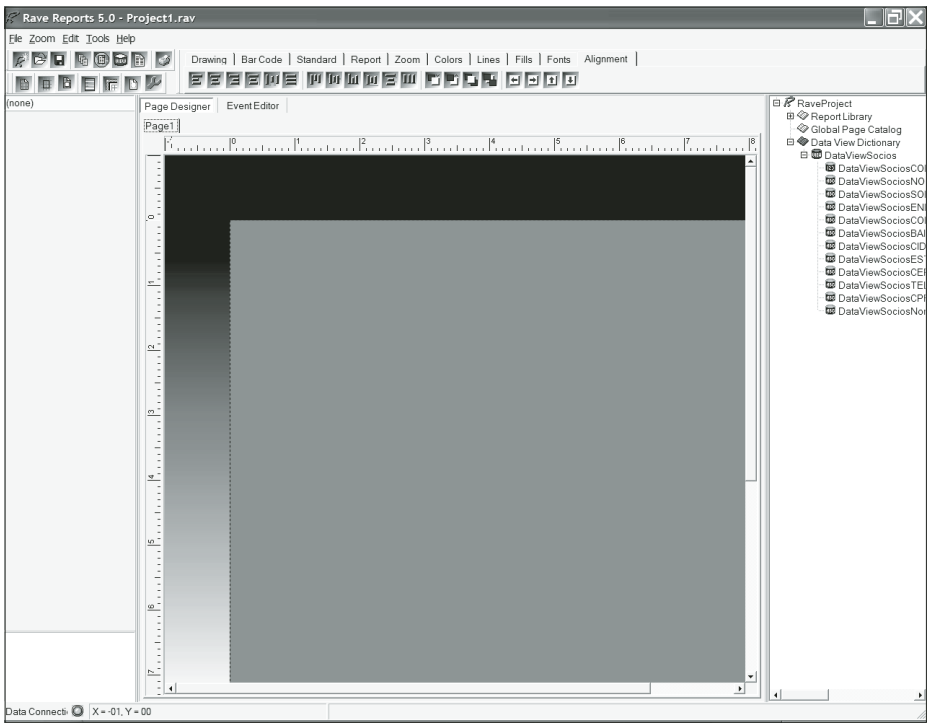


Figura 17.6: Incluindo um componente Region.

12. Inclua um componente Band (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) na área superior do componente Region1.

13. Selecione os três pontinhos à direita da propriedade BandStyle para exibir a caixa de diálogo Band Style Editor, mostrada na Figura 17.7.

14. Marque a opção Body Header e o botão Ok para fechar esta caixa de diálogo.

15. Altere para 0,990 o valor da propriedade Height.

16. Inclua nesta banda um componente Text (página Standard da paleta de componentes do ambiente de desenvolvimento do Rave Reports) e atribua os seguintes valores às suas principais propriedades:

    Name: Título
    Text: Relatório de Clientes
    Font: Arial, Bold Italic, sublinhada, tamanho 20.
    Top: 0,080

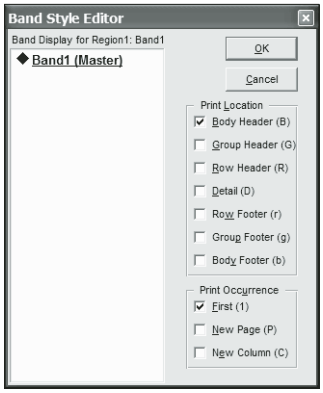


Figura 17.7: A caixa de diálogo Band Style Editor.

17. Centralize horizontalmente este componente na banda, selecionando o item Center Horizontally na página Alignment da paleta de componentes do ambiente de desenvolvimento do Rave Reports.

18. Coloque mais três componentes Text e defina a propriedade Text de cada um deles como Código, Nome e Sobrenome, respectivamente. Defina para estes componentes a mesma fonte e cuide de seu alinhamento. Seu relatório deverá ficar com o aspecto mostrado na figura a seguir.

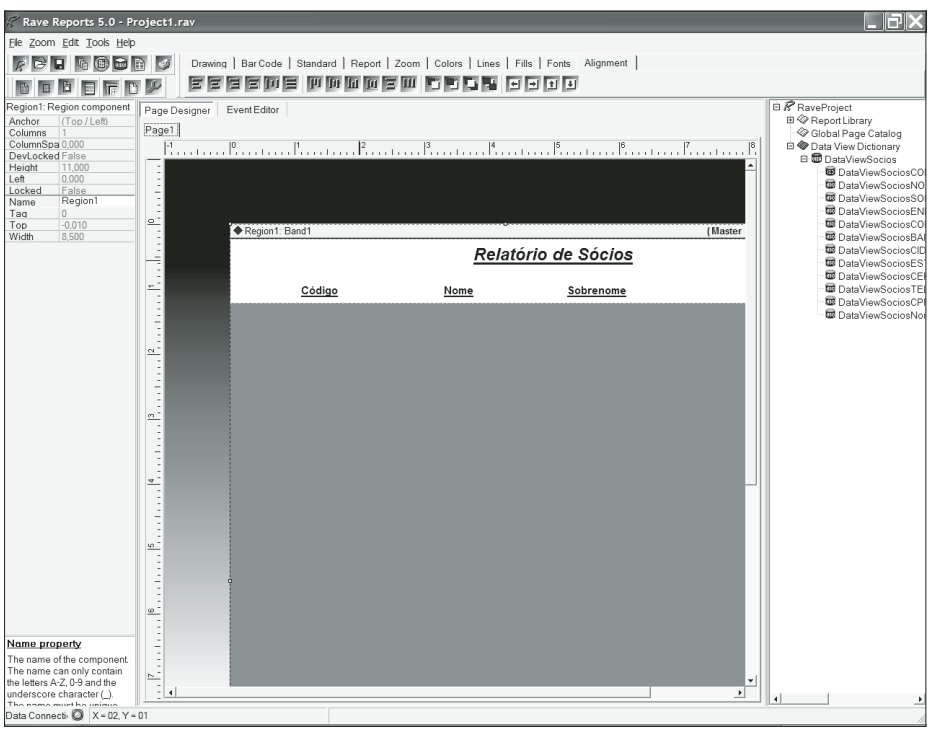


Figura 17.8: Inclusão de componentes do tipo Text.

19. Inclua um componente DataBand (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) abaixo da banda existente. Esta banda será a banda na qual serão exibidos os dados dos registros das tabelas.

20. Atribua o valor DataviewSocios à propriedade Dataview desta banda.
21. Nesta banda, coloque três componentes DataText (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) e atribua o valor DataviewSocios à propriedade Dataview de cada um destes componentes.
22. Configure a propriedade Datafield de cada um destes componentes como CODIGOSOCIO, NOME e SOBRENOME, respectivamente.
23. Selecione o item Save do menu File do ambiente de desenvolvimento do Rave Reports e salve o arquivo de projeto de relatório com o nome RelatClube, no mesmo diretório do seu projeto.
24. Para Pré-visualizar o relatório, selecione Execute Report do menu File do ambiente de desenvolvimento do Rave Reports. Será exibida a caixa de diálogo Output Options mostrada na figura a seguir.

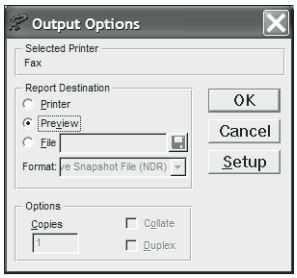


Figura 17.9: A caixa de diálogo Output Options.

25. Nesta caixa de diálogo, selecione a opção Preview e o botão Ok para Pré-visualizar o relatório, como mostrado na Figura 17.10.
26. Salve a unit associada ao Datamodule Relatórios com o nome UnitRelatorios, e inclua o nome desta unit na cláusula uses da unit do formulário principal da aplicação.
27. No Delphi 7, Atribua o valor RelatClube.rav à propriedade ProjectFile do componente RvClube.
28. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Sócios do menu Relatório, a ser compartilhado pelo item Sócios do submenu Relatórios do menu pop-up.

```
procedure TFormPrincipal.RelatorioSociosClick(Sender: TObject);
begin
  Relatorios.RvClube.SelectReport('RelatSocios',False);
  Relatorios.RvClube.Execute;
end;
```

A primeira linha de código seleciona o relatório a ser exibido (podemos ter vários relatórios relacionados a um único componente RvProject) e a segunda manda executar o relatório selecionado.

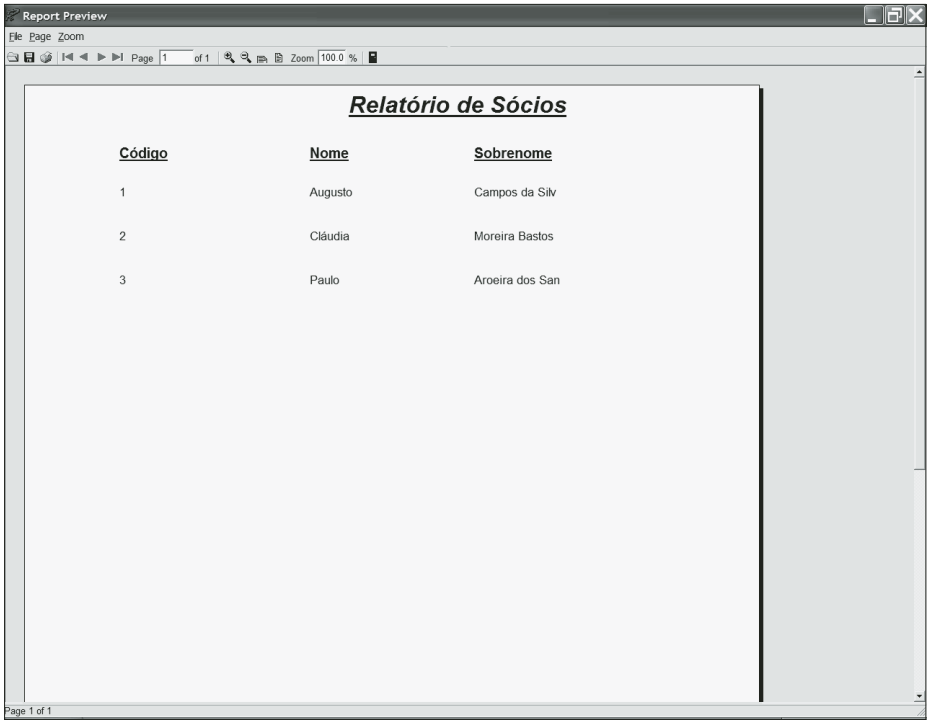


Figura 17.10: Pré-visualizando o Relatório.

Criando um Relatório de Atividades

Para criar um relatório com os dados das atividades, você deve executar os seguintes procedimentos:

1. Inclua no Datamodule Relatórios mais um componente RvDatasetConnection, e altere o valor da sua propriedade Name para RvDatasetConnectionSocios.
2. Atribua o valor Dados.SimpleDatasetAtividades à propriedade Dataset do componente RvDatasetConnectionAtividades.
3. Dê um duplo clique com o botão esquerdo do mouse sobre o componente RvClube. Será exibido o ambiente de desenvolvimento do Rave Reports.
4. Selecione o item New Report do menu File do ambiente do Rave Reports, para criar um novo relatório.
5. Selecione e expanda a opção Repot Library, no lado direito do ambiente do Rave Reports. Selecione o item Report2 e altere para RelatAtividades a sua propriedade Name.
6. Selecione o item New Data Object do menu File do ambiente do Rave Reports, para exibir a caixa de diálogo Data Connection.
7. Nesta caixa de diálogo, selecione o item Direct Data View e o botão Next. Serão exibidas as conexões de dados disponíveis.
8. Selecione a conexão correspondente à tabela de atividades.

9. Inclua um componente Region (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) na área da página do relatório (área central do ambiente de desenvolvimento do Rave Reports) e faça com que este componente ocupe toda a área disponível.
10. Inclua um componente Band (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) na área superior do componente Region1.
11. Selecione os três pontinhos à direita da propriedade BandStyle para exibir a caixa de diálogo Band Style Editor.
12. Marque a opção Body Header e o botão Ok para fechar esta caixa de diálogo.
13. Altere para 0,990 o valor da propriedade Height.
14. Inclua nesta banda um componente Text (página Standard da paleta de componentes do ambiente de desenvolvimento do Rave Reports) e atribua os seguintes valores às suas principais propriedades:

    Name: Título
    Text: Relatório de Atividades
    Font: Arial, Bold Italic, sublinhada, tamanho 20.
    Top: 0,080
15. Centralize horizontalmente este componente na banda, selecionando o item Center Horizontally na página Alignment da paleta de componentes do ambiente de desenvolvimento do Rave Reports.
16. Coloque mais três componentes Text e defina a propriedade Text de cada um deles como Código, Nome e Valor, respectivamente. Defina para estes componentes a mesma fonte e cuide de seu alinhamento.
17. Inclua um componente DataBand (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) abaixo da banda existente. Esta banda será a banda na qual serão exibidos os dados dos registros das tabelas.
18. Atribua o valor DataviewAtividades à propriedade Dataview desta banda.
19. Nesta banda, coloque três componentes DataText (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) e atribua o valor DataviewAtividades à propriedade Dataview de cada um destes componentes.
20. Configure a propriedade Datafield de cada um destes componentes como CODIGOATIVIDADE, NOME e VALOR, respectivamente.
21. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Atividades do menu Relatório, a ser compartilhado pelo item Atividades do submenu Relatórios do menu pop-up.

```
procedure TFormPrincipal.RelatorioAtividadesClick(Sender: TObject);
begin
  Relatorios.RvClube.SelectReport('RelatAtividades',False);
  Relatorios.RvClube.Execute;
end;
```

22. Selecione o item Save do menu File do ambiente de desenvolvimento do Rave Reports para salvar as alterações no projeto de relatório.

Criando um Relatório de Matrículas

Para criar um relatório com os dados das matrículas, você deve executar os seguintes procedimentos:

1. Inclua no Datamodule Relatórios mais um componente RvDatasetConnection, e altere o valor da sua propriedade Name para RvDatasetConnectionMatriculas.
2. Inclua a unit UnitExclusaoMatriculas na cláusula uses da unit associada ao Datamodule Relatórios.
3. Atribua o valor FormExclusaoMatriculas.SimpleDatasetConsulta à propriedade Dataset do componente RvDatasetConnectionMatriculas.
4. Dê um duplo clique com o botão esquerdo do mouse sobre o componente RvClube. Será exibido o ambiente de desenvolvimento do Rave Reports.
5. Selecione o item New Report do menu File do ambiente do Rave Reports, para criar um novo relatório.
6. Selecione e expanda a opção Report Library, no lado direito do ambiente do Rave Reports. Selecione o item Report3 e altere para RelatMatriculas a sua propriedade Name.
7. Selecione o item New Data Object do menu File do ambiente do Rave Reports, para exibir a caixa de diálogo Data Connection.
8. Nesta caixa de diálogo, selecione o item Direct Data View e o botão Next. Serão exibidas as conexões de dados disponíveis.
9. Selecione a conexão correspondente à tabela de Matrículas.
10. Inclua um componente Region (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) na área da página do relatório (área central do ambiente de desenvolvimento do Rave Reports) e faça com que este componente ocupe toda a área disponível.
11. Inclua um componente Band (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) na área superior do componente Region1.
12. Selecione os três pontinhos à direita da propriedade BandStyle para exibir a caixa de diálogo Band Style Editor.
13. Marque a opção Body Header e o botão Ok para fechar esta caixa de diálogo.
14. Altere para 0,990 o valor da propriedade Height.
15. Inclua nesta banda um componente Text (página Standard da paleta de componentes do ambiente de desenvolvimento do Rave Reports) e atribua os seguintes valores às suas principais propriedades:

    Name: Título
    Text: Relatório de Matrículas
    Font: Arial, Bold Italic, sublinhada, tamanho 20.
    Top: 0,080

16. Centralize horizontalmente este componente na banda, selecionando o item Center Horizontally na página Alignment da paleta de componentes do ambiente de desenvolvimento do Rave Reports.

17. Coloque mais três componentes Text e defina a propriedade Text de cada um deles como Nome, Sobrenome e Atividade, respectivamente. Defina para estes componentes a mesma fonte e cuide de seu alinhamento.
18. Inclua um componente DataBand (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) abaixo da banda existente. Esta banda será a banda na qual serão exibidos os dados dos registros das tabelas.
19. Atribua o valor DataviewMatriculas à propriedade Dataview desta banda.
20. Nesta banda, coloque três componentes DataText (página Report da paleta de componentes do ambiente de desenvolvimento do Rave Reports) e atribua o valor DataviewMatriculas à propriedade Dataview de cada um destes componentes.
21. Configure a propriedade Datafield de cada um destes componentes como NOME, SOBRENOME e ATIVIDADE, respectivamente.
22. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Matrículas do menu Relatório, a ser compartilhado pelo item Matrículas do submenu Relatórios do menu pop-up.

```
procedure TFormPrincipal.RelatorioMatriculasClick(Sender: TObject);
begin
  Relatorios.RvClube.SelectReport('RelatMatrículas',False);
  Relatorios.RvClube.Execute;
end;
```

23. Selecione o item Save do menu File do ambiente de desenvolvimento do Rave Reports para salvar as alterações no projeto de relatório.

Capítulo 18

Incorporando o Recurso de Help On-Line à Nossa Aplicação

Neste capítulo serão apresentados os procedimentos necessários à incorporação do recurso de Help On-Line ao seu aplicativo.

Know-How em: Criação de Arquivos de Help

PRÉ-REQUISITOS

- Noções básicas da utilização do ambiente de desenvolvimento do Borland Delphi 7.
- Conceitos básicos sobre a utilização de utilitários para o ambiente Windows.

METODOLOGIA

- Apresentação e descrição dos principais conceitos relacionados à criação de Arquivos de Help com o Word e o Microsoft Help Compiler.

TÉCNICA

- Apresentação e descrição dos procedimentos necessários à criação de Arquivos de Help com o Word e o Microsoft Help Compiler.

Criando um Arquivo de Help

Para criar um arquivo de Help você deve usar um editor capaz de manipular arquivos no formato RTF, como por exemplo o Word for Windows. Neste capítulo vamos usar o Word XP para criar o nosso arquivo de Help.

Definindo uma Página de Índice

A maioria dos arquivos de Help On-Line apresenta uma página de índice, onde são mostrados os títulos das principais seções ou tópicos do arquivo. No nosso caso, apenas para exemplificar, vamos criar uma página com o título "Índice do Programa de Cadastro dos Sócios do Clube" e os seguintes tópicos:

- Cadastrando um Novo Sócio
- Alterando os Dados de um Sócio
- Excluindo um Sócio
- Consultando um Sócio
- Cadastrando uma Nova Atividade
- Alterando os Dados de uma Atividade
- Excluindo uma Atividade
- Consultando uma Atividade
- Efetuando uma Matrícula
- Removendo uma Matrícula
- Consultando Matrículas
- Relatórios

A figura a seguir mostra o aspecto da página criada, usando o Microsoft Word 2000.

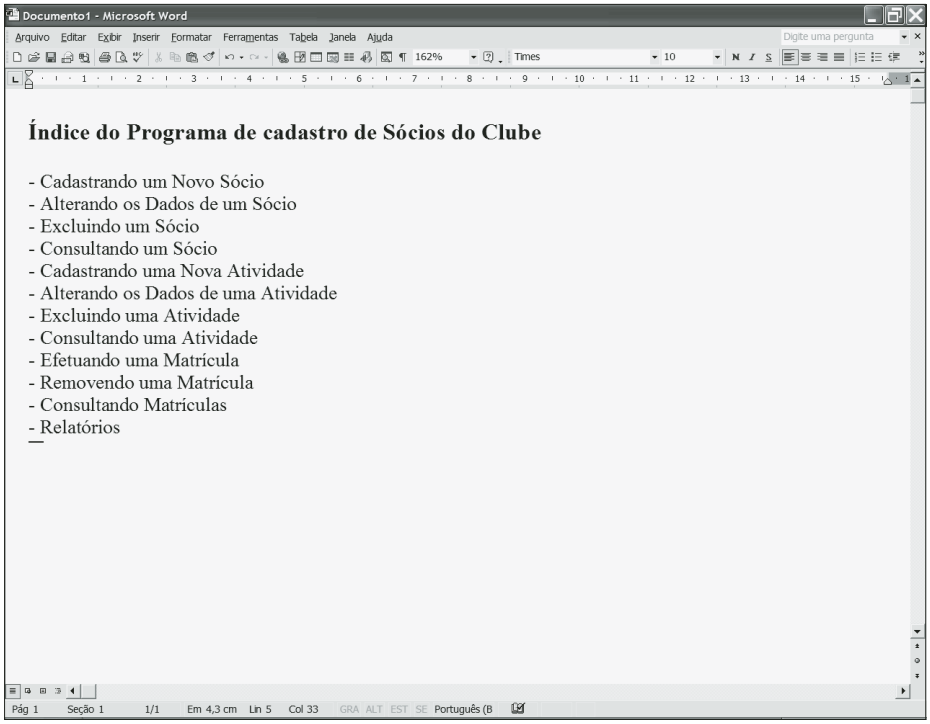


Figura 18.1: Criando a página de tópicos.

CRIANDO UM ARQUIVO RTF

Para criar um arquivo RTF, você deve executar os seguintes procedimentos:

1. Criar um documento no Word, como a página de tópicos mostrada anteriormente.
2. Selecione o item Salvar Como, do menu Arquivo. Será exibida a caixa de diálogo Salvar como, mostrada na figura a seguir.

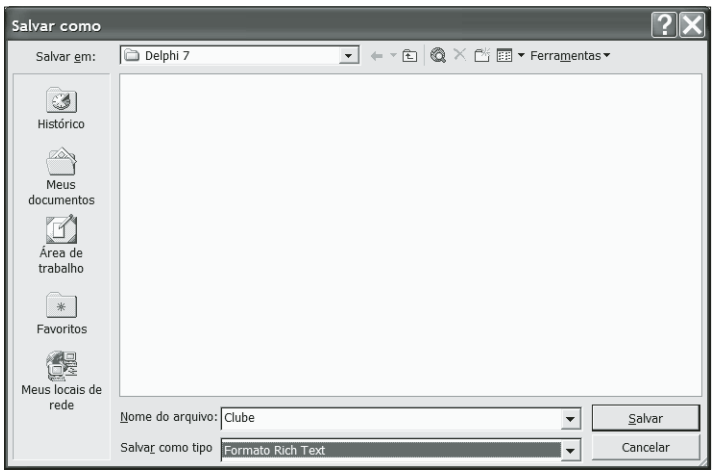


Figura 18.2: Salvando um arquivo no formato rtf.

3. Nesta caixa de diálogo digite o nome do arquivo e selecione a opção Rich Text Format (rtf) na caixa combo "Salvar como tipo:".

Criando uma Página Para Cada Tópico

Cada tópico deve ser digitado em uma página distinta, como mostra a figura a seguir.

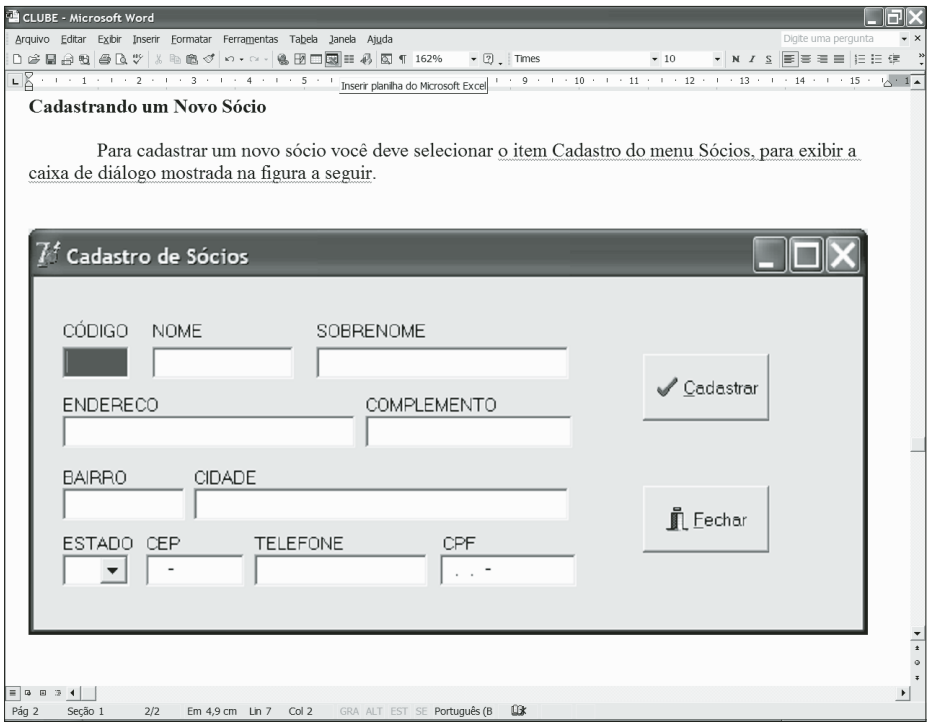


Figura 18.3: Definindo uma página para cada tópico.

Criando Strings de Contexto

Cada tópico de um arquivo de Help deve ser identificado por uma string de contexto. É o valor definido para esta string de contexto que será usado para que este tópico seja acessado a partir de outro. Vamos definir "Cad_Socios" como a string de contexto para o tópico "Cadastrando um Novo Sócio".

Para criar uma string de contexto, você deve executar os seguintes procedimentos:

1. Posicione o cursor no início da página que representa o tópico cuja string de contexto está sendo definida.
2. Selecione o item Notas... do submenu Referência do menu Inserir do Word XP. Será exibida a caixa de diálogo "Nota de Rodapé e Nota de Fim", mostrada na próxima figura.

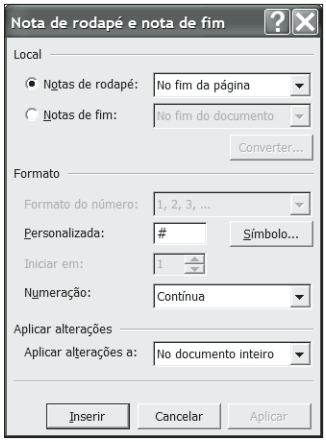


Figura 18.4: A caixa de diálogo Nota de rodapé e nota de fim.

3. Nesta caixa de diálogo, selecione a opção Notas de rodapé.
4. Digite o caractere # na caixa de texto à direita da opção Personalizada.
5. Selecione o botão OK, para fechar a caixa de diálogo.
6. Digite a string de contexto na nota de rodapé, logo após o sinal "#", criado no passo anterior, como mostra a figura a seguir.

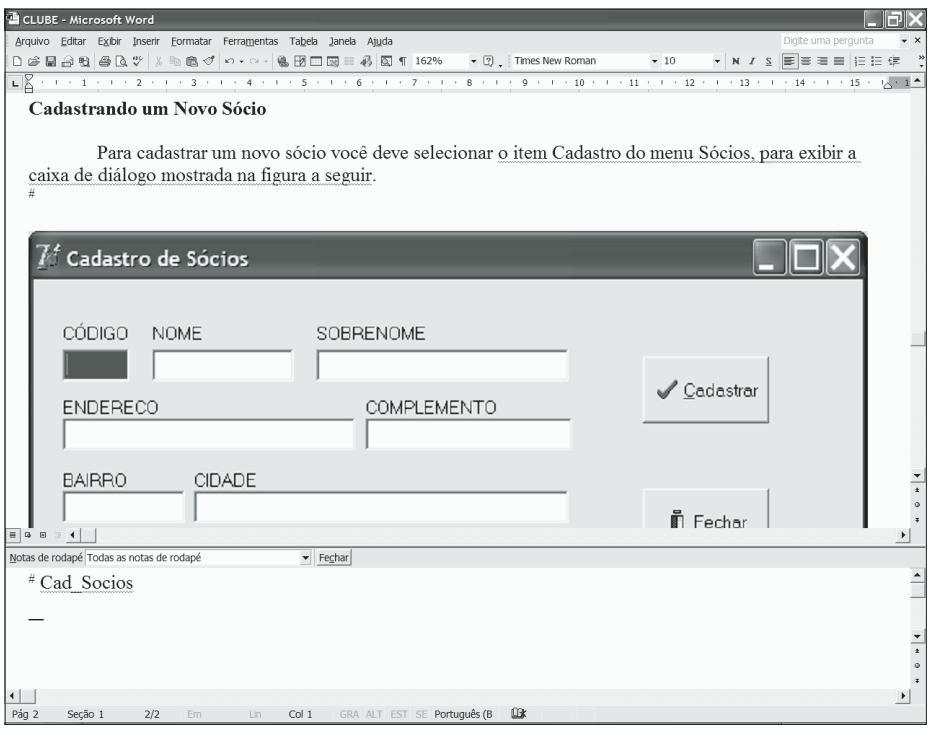


Figura 18.5: Definindo uma string de contexto para um tópico.

Uma string de contexto não pode conter espaços em branco e deve ter, no máximo, 255 caracteres.

Criando uma Palavra-chave Para um Tópico

Cada tópico em um arquivo de Help deve possuir uma palavra-chave (que pode ser um texto com espaços em branco), que é a palavra pela qual o tópico poderá ser pesquisado pelo Help do Windows. O processo de criação de uma palavra-chave é idêntico ao de criação de uma string de contexto, exceto que o caractere a ser digitado na caixa de texto à direita da opção Personalizada, na caixa de diálogo "Nota de rodapé e nota de fim", será um "K" ao invés de um "#". Normalmente se escolhe um texto significativo, que neste caso pode ser "Cadastro de Sócios".

Se você executar os passos descritos no tópico anterior para a criação desta palavra-chave, a sua página de tópico deverá ficar com o aspecto mostrado na figura a seguir.

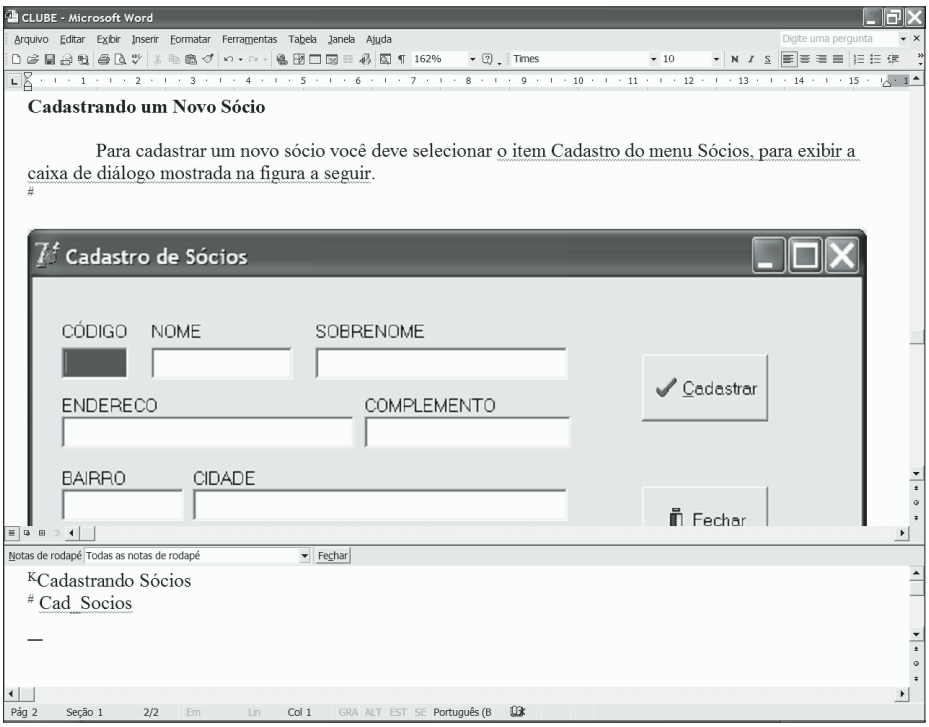


Figura 18.6: Definindo uma palavra-chave para um tópico.

Criando um Título Para um Tópico

Cada tópico de um arquivo de Help deve possuir um título pelo qual o tópico será referenciado pelo Help do Windows, quando selecionada a sua palavra-chave após uma pesquisa pelo Help. O processo de criação de um título é idêntico ao de criação de uma string de contexto, exceto que o caractere a ser digitado na caixa de texto à direita da opção Personalizada na caixa de diálogo "Nota de rodapé e nota

de fim" será um "$" ao invés de um "#". Normalmente se escolhe um texto significativo, que neste caso pode ser "Cadastrando Novos Sócios".

Se você executar os passos descritos nos tópicos anteriores para a criação deste título, a sua página de tópico deverá ficar com o aspecto mostrado na próxima figura.

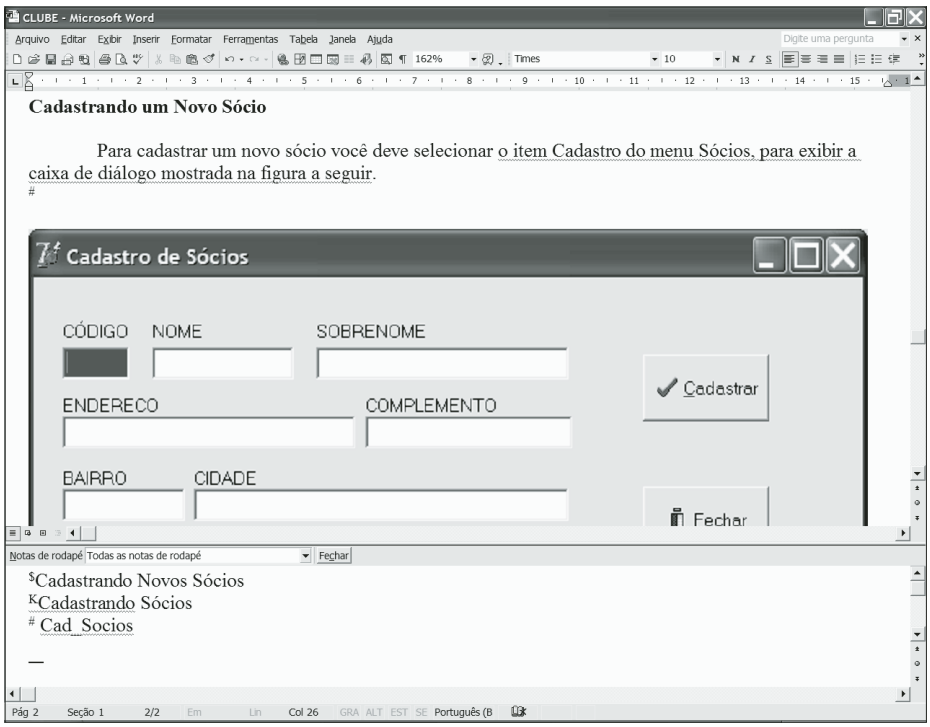


Figura 18.7: Definindo um título para um tópico.

Associando um Número de Página a um Tópico

Cada tópico de um arquivo de Help deve possuir um número de página, que estabelece a sua localização numa seqüência de páginas a serem percorridas pelo usuário com os botões << e >> no Help do Windows. O processo de criação de um número de página é idêntico ao de criação de uma string de contexto, exceto que o caractere a ser digitado na caixa de texto à direita da opção Personalizada na caixa de diálogo "Nota de rodapé e nota de fim" será um "+" ao invés de um "#". O número de página pode ser precedido por uma seqüência de caracteres que define um grupo de páginas, desde que terminado por dois-pontos (:).

Se você executar os passos descritos nos tópicos anteriores para a criação do número de página Topicos:001, a sua página de tópico deverá ficar com o aspecto mostrado na Figura 18.8.

Crie você mesmo as demais páginas de tópicos, executando os procedimentos descritos anteriormente.

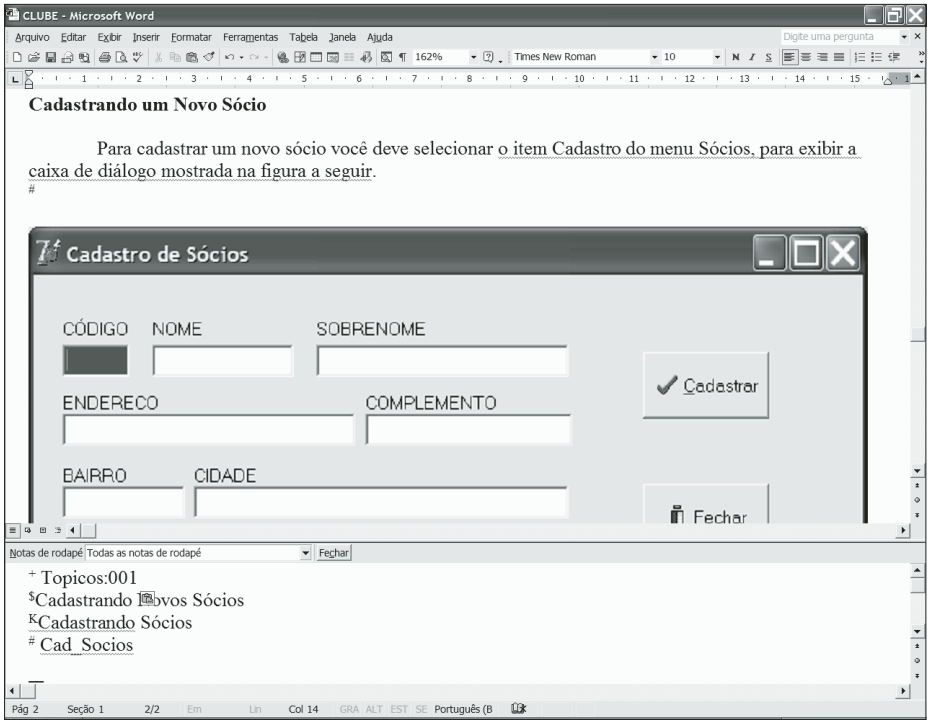


Figura 18.8: Definindo um título para um tópico.

Estabelecendo a Conexão Entre Tópicos

Se você já utilizou o arquivo de Help das principais aplicações desenvolvidas para o ambiente Windows, já deve ter percebido que algumas palavras são exibidas sublinhadas e na cor verde. Quando você seleciona estas palavras com o mouse, ocorre um salto para um outro tópico de um arquivo de Help.

Para implementar o salto para a página de tópico Cadastrando um Novo Sócio. quando o usuário selecionar este nome na página de índice, basta executar os seguintes procedimentos:

1. Selecione o texto Cadastrando um Novo Sócio na página de índice.
2. Aplique o formato de duplo sublinhado ao texto selecionado no passo anterior.
3. Imediatamente após o texto selecionado e formatado no passo anterior, digite a string de contexto do tópico para onde ocorrerá o "salto", selecione-a e aplique o formato de texto escondido. Sua página de índices deverá ficar com o aspecto mostrado na próxima figura.

Crie você mesmo as conexões para os demais tópicos, executando os passos descritos anteriormente.

Pronto! Está estabelecida a conexão entre tópicos do Help!

Caso você não esteja enxergando o texto oculto como mostra a Figura 18.9, marque a opção Texto Oculto na guia Exibir da caixa de diálogo Opções do Word XP, exibida quando se seleciona o item Opções, do menu Ferramentas (Figura 18.10).

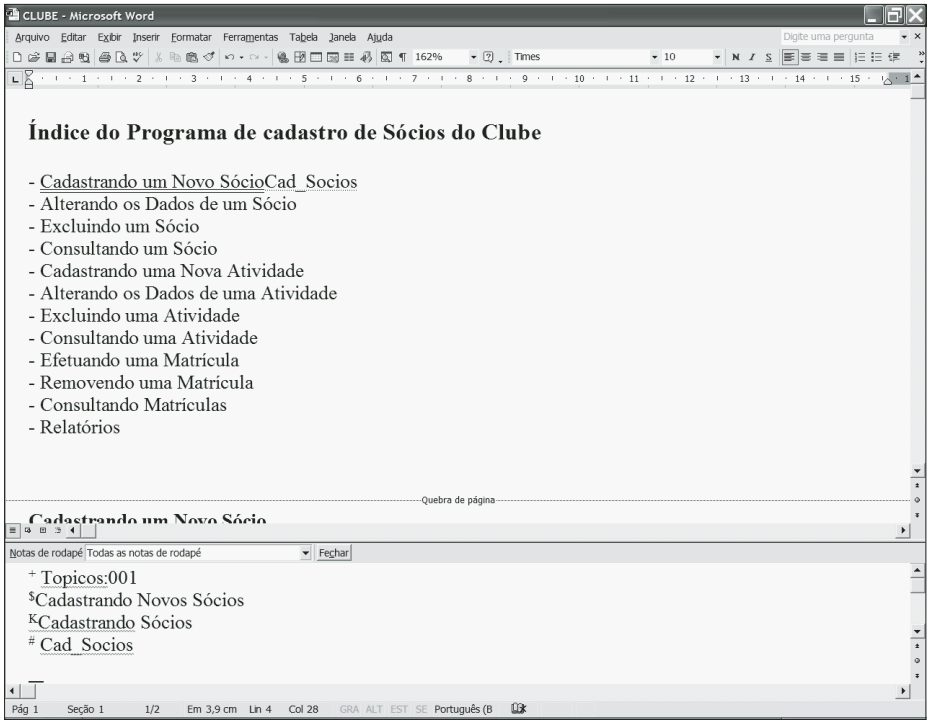


Figura 18.9: Definindo uma conexão para um tópico.

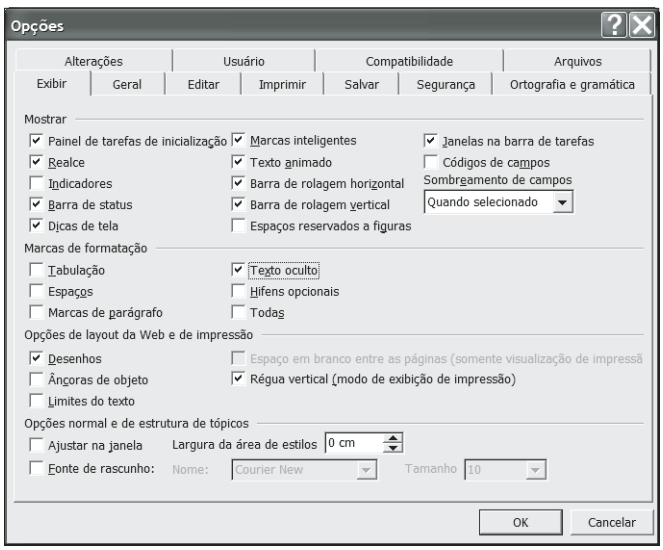


Figura 18.10: A página Exibir da caixa de diálogo Opções do Word XP.

Crie as demais páginas de tópicos e defina as strings de auxílio, palavras-chave, título e numeração como indicado. Defina a numeração da página de índice como Tópico:001.

Criando um Arquivo de Projeto de Help

Além do arquivo com a extensão rtf, você precisa criar um arquivo de projeto (Help Project File), que armazenará as informações a serem usadas pelo compilador de arquivos de Help. Este arquivo deve ter a extensão HPJ (não será criado no Word) e o seu nome será o nome do arquivo de Help, tendo este último a extensão EXE.

Um arquivo de projeto de Help é composto de várias seções, descritas a seguir:

```
A seção OPTIONS possui a seguinte sintaxe:
[OPTIONS]
CONTENTS = string_de_contexto
TITLE = title
COMPRESS = nível_de_compressão
ERRORLOG = Nome_do_arquivo_de_descrição_de_erros
```

A primeira linha define a string de contexto do tópico principal do arquivo de Help (na realidade, o Help On-Line de uma aplicação pode ser composto por vários arquivos). Se esta linha for omitida, o primeiro tópico do primeiro arquivo de ajuda será assumido como tópico principal.

A segunda linha define o texto exibido na barra de títulos do arquivo de Help.

A terceira linha define o tipo de compressão do arquivo. O nível de compressão pode assumir um dos valores definidos na tabela a seguir:

Valor	Velocidade de Compilação	Tamanho do arquivo
FALSE	Rápida	Grande (sem compressão)
MEDIUM	Média	Médio (compressão média)
HIGH	Lenta	Pequeno (compressão alta)
0	Rápida	Grande (sem compressão)
1	Lenta	Pequeno (compressão alta)
NO	Rápida	Grande (sem compressão)
TRUE	Lenta	Pequeno (compressão alta)
YES	Lenta	Pequeno (compressão alta)

Se esta linha for omitida, não haverá compressão.

A quarta linha define o arquivo no qual serão gravadas as mensagens de erros de compilação. Se esta linha for omitida, as mensagens de erro serão exibidas na tela mas não serão enviadas para nenhum arquivo.

A seção CONFIG possui a seguinte sintaxe:

```
[CONFIG]
BrowseButtons()
```

Esta linha, se incluída, faz com que os botões << e >> sejam exibidos, permitindo ao usuário percorrer uma seqüência de páginas do arquivo de Help.

A seção FILES possui a seguinte sintaxe:

```
[FILES]
arquivo_1 com extensão rtf.
.............
arquivo_n com extensão rtf.
```

Nesta seção são listados os arquivos que vão compor o arquivo de Help.

Você pode criar este arquivo em um editor de texto ou usando o Microsoft Help Workshop, como mostrado no próximo tópico.

CRIANDO E COMPILANDO O ARQUIVO DE PROJETO DE HELP COM O MICROSOFT HELP WORKSHOP

Para compilar o arquivo de Help com o Microsoft Help Workshop, você deve executar os seguintes procedimentos:

1. Execute o programa HCW.EXE, situado no subdiretório c:\Arquivos de Programas\Borland\\ Delphi 7\Help\Tools. Será exibida a seguinte tela de abertura:

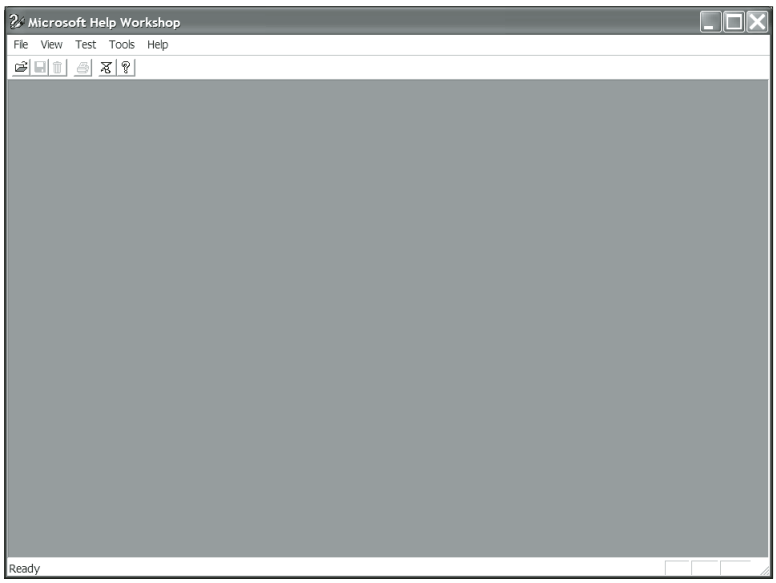


Figura 18.11: A tela de abertura do Microsoft Help Workshop.

2. Selecione o item New do menu File. Será exibida a caixa de diálogo New (Figura 18.12).
3. Selecione a opção Help Project.
4. Selecione o botão OK, para fechar a caixa de diálogo. Será exibida a caixa de diálogo Project File Name (Figura 18.13).

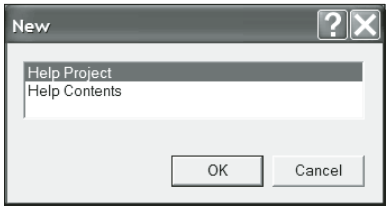


Figura 18.12: A caixa de diálogo New.

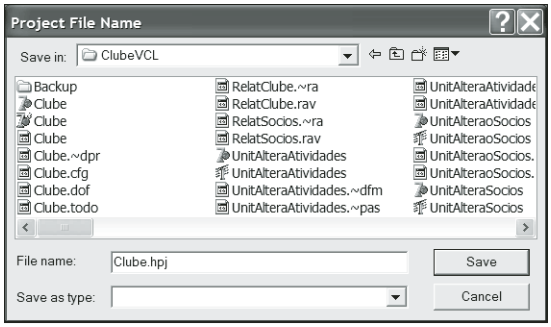


Figura 18.13: A Caixa de diálogo Project File Name.

5. Digite o nome do arquivo de projeto.
6. Selecione o botão Salvar, para fechar a caixa de diálogo. Será criado o esqueleto do arquivo de projeto, como mostra a Figura 18.14.

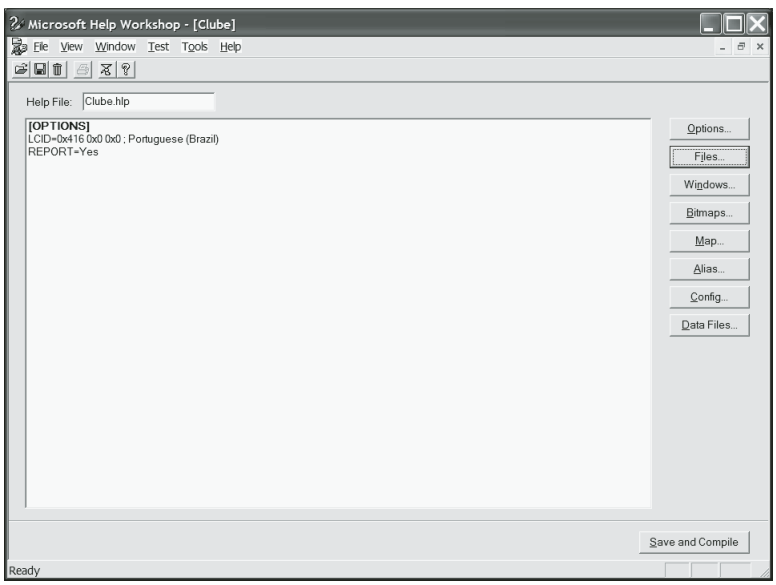


Figura 18.14: Criando o arquivo de projeto.

7. Selecione o botão Files, para exibir a caixa de diálogo Topic Files (Figura 18.15).

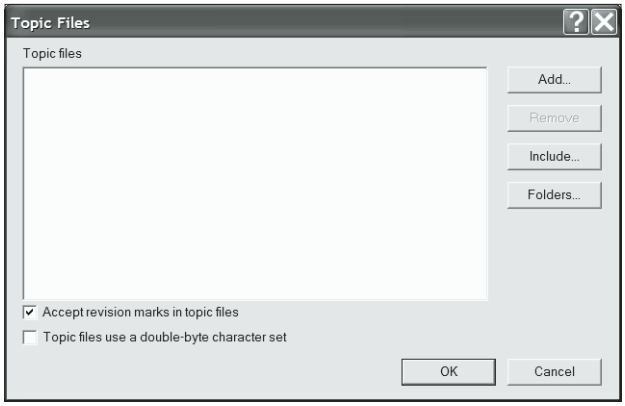


Figura 18.15: A caixa de diálogo Topic Files.

8. Selecione o botão Add, para exibir a caixa de diálogo Abrir (Figura 18.16).

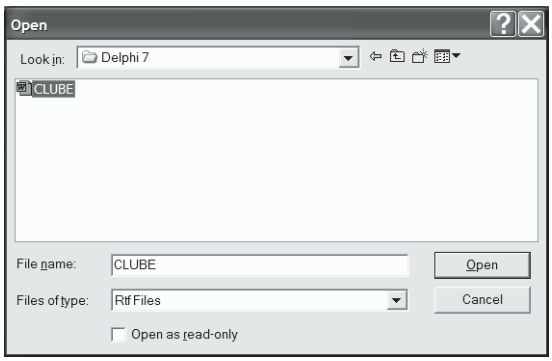


Figura 18.16: A caixa de diálogo Abrir.

9. Selecione o arquivo desejado.
10. Clique em Open para fechar a caixa de diálogo Abrir.
11. Clique em OK para fechar a caixa de diálogo Topic Files. Seu arquivo será incluído no projeto, como mostra a figura a seguir.
12. Selecione o botão Save and Compile, para salvar e compilar o arquivo de projeto de Help.

Pronto! Você acaba de criar o arquivo Clube.hlp (o arquivo de help do seu aplicativo).

Você pode configurar as diversas seções do arquivo de projeto de Help selecionando o botão correspondente na janela principal do Microsoft Help Workshop.

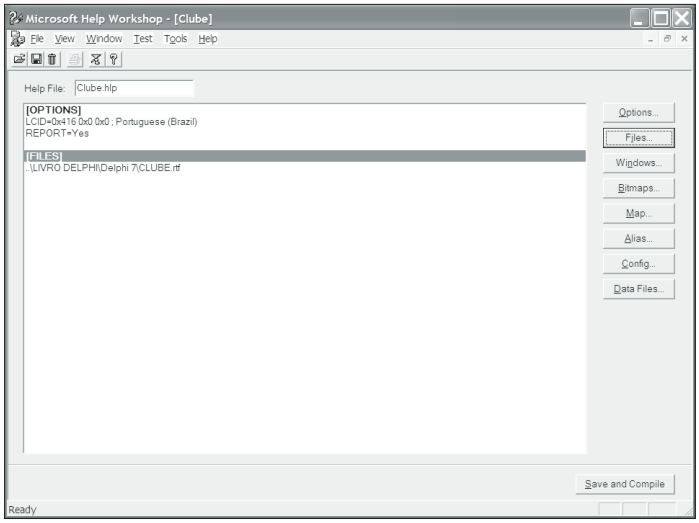


Figura 18.17: Adição do arquivo Clube.rtf no arquivo de projeto.

Associando o Arquivo de Help à sua Aplicação

Para criar um Help sensível ao contexto, execute os seguintes procedimentos a partir do ambiente do Delphi 7:

1. Abra o seu projeto de aplicação.
2. Selecione o item Options do menu Project. Será exibida a caixa de diálogo Project Options.
3. Selecione a página Application na caixa de diálogo Project Options.
4. Digite o path e o nome do arquivo de Help na caixa de texto Help file, como mostra a Figura 18.18.

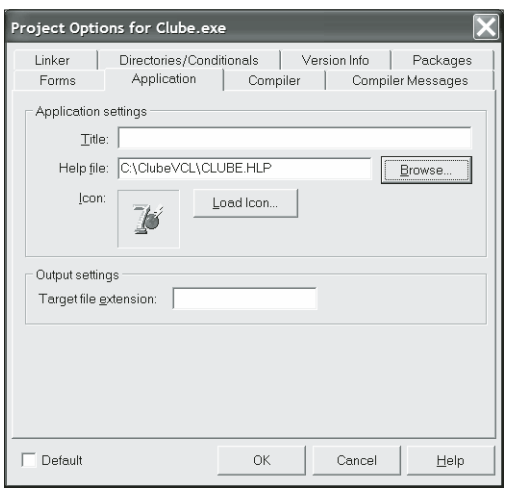


Figura 18.18: A caixa de diálogo Project Options.

5. Selecione o botão OK para fechar a caixa de diálogo Project Options.

Associando um Componente a um Tópico do Arquivo de Help

Cada componente do Borland Delphi 7 possui uma propriedade chamada HelpContext, à qual deve ser atribuído um valor inteiro. Para associar um tópico a um componente, deve-se atribuir um valor inteiro (001, por exemplo) e criar uma correspondência entre a string de contexto e este número na seção [MAP] do arquivo de projeto (esta seção ainda não havia sido mencionada).

```
Exemplo:
[MAP]
Cad_Socios   002
```

Para associar um número a uma string de contexto, você deve executar os seguintes procedimentos:

1. Abra o arquivo de projeto, selecionando o item Open no menu File do Microsoft Help Workshop.
2. Selecione o botão Map. Será exibida a caixa de diálogo Map. Selecione o botão Add para exibir a caixa de diálogo Add Map Entry (Figura 18.19).

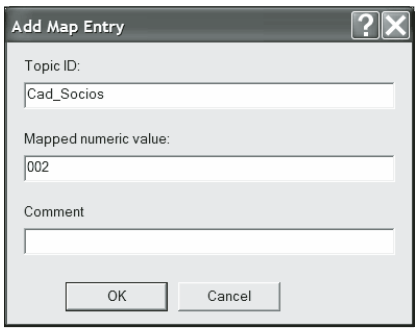


Figura 18.19: A caixa de diálogo Add Map Entry.

3. Digite os valores adequados nas caixas de texto Topic ID e Mapped numeric value.
4. Selecione o botão OK para fechar a caixa de diálogo. Será exibida a caixa de diálogo Map, mostrada na figura a seguir.

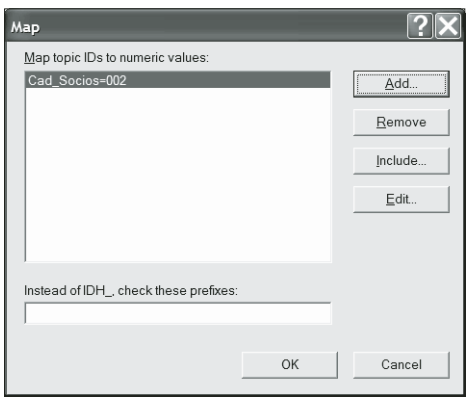


Figura 18.20: A caixa de diálogo Map.

5. Selecione o botão Save and Compile, para salvar e recompilar o arquivo de projeto de Help.
6. Atribua o valor 002 à propriedade HelpContext do componente desejado.

Exibindo o Arquivo de Help em Resposta a um Item de Menu

Para que o arquivo de Help seja exibido quando o usuário selecionar o item Ajuda, do menu Help, ou pressionar a tecla de função F1, basta executar os seguintes procedimentos:

1. Incluir a seguinte linha de código no procedimento associado ao evento OnClick do item de menu:

```
WinHelp(FormPrincipal.handle,'C:\loja\loja.hlp',HELP_CONTENTS,0);
```

2. Atribuir o valor F1 à propriedade ShortCut do item de menu.

A função WinHelp é uma função da API do Windows que requer os seguintes parâmetros:

- O handle da janela.
- Uma string com o path completo e o nome do arquivo de Help.
- O tipo de Help.
- Um inteiro (geralmente pode ser usado o valor 0).

Assim, quando o usuário selecionar o item Ajuda, do menu Help, ou pressionar a tecla de função F1, será exibida a tela do arquivo de Help associado à aplicação.

Capítulo 19

Programação Orientada a Objetos em Delphi 7

Nos últimos anos, as técnicas de programação orientada a objetos vêm ocupando lugar de destaque no cenário de desenvolvimento de aplicações, principalmente após o surgimento de novas linguagens que passaram a incorporar esse recurso, como o C++ e o Object Pascal (a linguagem de programação utilizada pelo Delphi 7).

Inicialmente, é importante salientar que as técnicas de programação orientada a objetos não constituem apenas um novo estilo de codificação, mas uma nova maneira de se abordar um problema a ser resolvido com o uso do computador.

As linguagens de programação procedurais, predominantes anteriormente, tratavam os dados de um problema e os procedimentos que manipulavam esses dados como entidades distintas. Geralmente os dados eram passados como parâmetros a estes procedimentos (a menos que fossem definidos como variáveis globais). Era nítida, portanto, a separação entre dados e funções como entidades distintas.

As linguagens de programação orientadas a objetos, por outro lado, têm como característica principal a capacidade de reunir os dados e as funções que operam sobre estes dados em uma nova e única entidade – denominada classe. Desta maneira, as funções e os procedimentos (denominados métodos no linguajar da Programação Orientada a Objetos) definidos internamente à classe passam a ter a capacidade de acessar diretamente estes dados, que não precisam mais ser passados como parâmetros.

Esta abordagem representa uma visão muito mais próxima da realidade, uma vez que, no nosso cotidiano, lidamos o tempo todo com objetos que reúnem características (as propriedades que os definem) e funcionalidades (caracterizadas pelos seus métodos) em uma única entidade.

Este capítulo trata dos princípios da filosofia da programação orientada a objetos, comparando-os aos da programação estruturada, tradicionalmente utilizada no desenvolvimento de programas por meio do Pascal Procedural.

Inicialmente abordamos a programação estruturada em Object Pascal, para posteriormente apresentar ao leitor as características da programação orientada a objetos, de forma a tornar mais claras as diferenças entre as duas abordagens

É evidente que não pretendemos esgotar este assunto, pois existem livros inteiramente dedicados à análise e programação orientada a objetos, mas pretende-se dar ao leitor os conhecimentos necessários ao desenvolvimento de sistemas em Object Pascal (a linguagem de programação orientada a objetos do Delphi).

KNOW-HOW em: Programação Procedural em Linguagem Object Pascal

PRÉ-REQUISITOS

- Experiência em programação estruturada com versões anteriores da linguagem Pascal.

METODOLOGIA

- Apresentação do problema: Criação de um programa que utilize o Pascal procedural utilizando-se o ambiente de desenvolvimento integrado do Delphi 7.

TÉCNICA

- Utilização do ambiente de desenvolvimento integrado do Delphi 7 para criar aplicações tipo Console.

Apresentação do Problema

Este tópico se destina a apresentar os procedimentos necessários ao desenvolvimento de aplicações tipo console com o Delphi 7, e sua inclusão decorre da necessidade de apresentarmos, nos tópicos que se seguem, os fundamentos da programação orientada a objetos sem a necessidade de nos referirmos aos elementos de criação da interface gráfica de uma aplicação.

Além disso, gostaria de iniciar este capítulo relatando um fato bastante interessante, mas que pode parecer um tanto estranho para os analistas, programadores e demais profissionais de informática.

Nos últimos anos, minha experiência acadêmica e profissional tem demonstrado que, apesar de toda a evolução dos sistemas operacionais com interface gráfica (como o Windows 95/98/Me/2000/NT/XP, por exemplo), os currículos escolares de muitas faculdades e universidades brasileiras têm evitado a inclusão de disciplinas que abordem exclusivamente a utilização de ambientes e linguagens visuais de programação, como o Delphi, o C++ Builder, o Visual Basic, etc.

Esse fato se deve, entre outras causas, aos seguintes fatores:

- Muitos professores ainda não aceitaram ou assimilaram os novos conceitos que envolvem as técnicas de programação orientada a objetos e a eventos.
- As linguagens visuais de programação, ao simplificarem demasiadamente os procedimentos necessários à criação da interface com o usuário, muitas vezes fazem com que o aluno diminua o seu interesse pelas importantíssimas técnicas de codificação (lógica, algoritmos, análise, etc.), colocando-as em segundo plano.

É importante salientar, entretanto, que as linguagens visuais de programação foram desenvolvidas com o objetivo de minimizar o tempo gasto no desenvolvimento dessa interface, justamente para que o analista ou programador dedique a maior parte do seu tempo às tarefas de codificação e implementação relacionadas à solução de um determinado problema (que é o objetivo de qualquer programa – automatizar a solução de um dado problema).

Há pouco tempo, antes do surgimento destas linguagens visuais, o desenvolvimento de um programa extremamente simples para o ambiente Windows – formado por uma única janela – requeria cerca de 80 linhas de código-fonte escrito em linguagem C. A criação da interface, então, exigia um enorme esforço de programação.

Conseqüentemente, as linguagens de programação procedurais (como o C e o Pascal) têm predominado nos currículos universitários (principalmente nos primeiros anos do curso).

É importante lembrar, no entanto, que a linguagem Object Pascal é, na realidade, uma evolução da linguagem Pascal e que o ambiente de desenvolvimento do Delphi 7 permite a codificação e a compilação de programas desenvolvidos utilizando-se o Pascal como linguagem de programação no estilo procedural.

O objetivo deste tópico será, portanto, mostrar como codificar e compilar programas escritos em linguagem Pascal, usando o ambiente de desenvolvimento do Delphi 7. Nos tópicos subseqüentes, ao

abordarmos a filosofia da programação orientada a objetos, as diferenças entre estas duas abordagens ficarão muito mais claras.

Utilizando o Ambiente do Delphi 7 Para o Pascal Procedural

Neste tópico serão apresentados os procedimentos necessários à utilização do ambiente de desenvolvimento do Delphi 7 para a criação de aplicações que utilizem o Pascal como linguagem de programação procedural. Esses tipos de aplicações não terão formulários (janelas), botões de comando ou qualquer outro tipo de componente da interface padrão do ambiente Windows, sendo definidas pelo Delphi 7 como aplicações do tipo console (serão executadas em uma janela padrão do DOS).

Para criar uma aplicação do tipo console, proceda da seguinte maneira:

1. Selecione o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items.
2. Selecione o item Console Application na página New desta caixa de diálogo, conforme indicado na figura a seguir.
3. Selecione o botão Ok, para fechar esta caixa de diálogo.

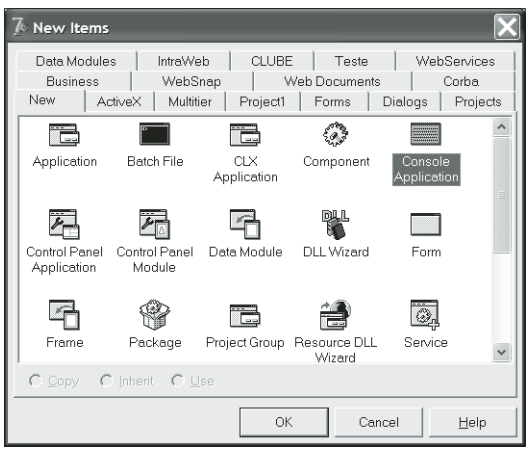


Figura 19.1: Criando uma aplicação tipo console com o Delphi.

A aplicação tipo console será criada com um único arquivo (arquivo de projeto, com extensão dpr), e cujo código gerado é reproduzido a seguir.

```
program Project1;
{$APPTYPE CONSOLE}
uses sysutils;
begin
  end.
```

4. Salve o seu projeto com o nome Console.dpr, usando o Ítem Save Project As do menu File.
5. O código-fonte do seu arquivo de projeto deverá ficar como indicado a seguir.

```
program console;
```

```
program Console;

{$APPTYPE CONSOLE}

uses sysutils;

begin

end.
```

No caso de um programa desenvolvido com o Pascal padrão, o início do programa é definido pela palavra-chave program, seguida pelo nome do programa e pelas instruções delimitadas entre o primeiro begin e o último end (que é acompanhado de um ponto, e não de um ponto-e-vírgula, pois encerra a unit).

Apenas para exemplificar o funcionamento de uma aplicação do tipo console, redefina o código do seu programa da maneira indicada a seguir.

```
program Console;

{$APPTYPE CONSOLE}

uses sysutils;

begin
     Writeln('Exemplo de Aplicativo Tipo Console');
     Readln;
end.
```

Execute esta aplicação, e verá a janela mostrada na figura a seguir.

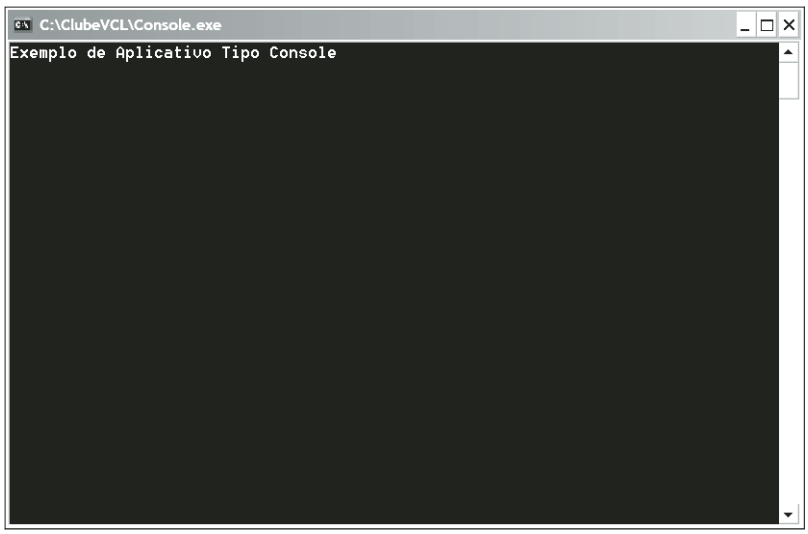


Figura 19.2: Executando uma aplicação tipo console com o Delphi 7.

É importante salientar, no entanto, que no caso do Delphi 7, este tipo de aplicação é de 32 bits do Windows (e não uma aplicação de 16 bits do DOS), e conseqüentemente pode chamar qualquer função definida pela API do Windows 95/98/Me/2000/NT/XP.

Experimente, por exemplo, redefinir este programa da maneira mostrada a seguir.

```
program Console;

{$APPTYPE CONSOLE}

uses
 Dialogs;
begin
     ShowMessage('Exemplo de Aplicação Tipo Console');
end.
```

A função ShowMessage é típica do ambiente Windows, mostrando que, embora seja do tipo Console, não é uma aplicação DOS (embora possua a "cara" e interface do DOS).

Execute esta aplicação, e verá a janela mostrada na figura a seguir.

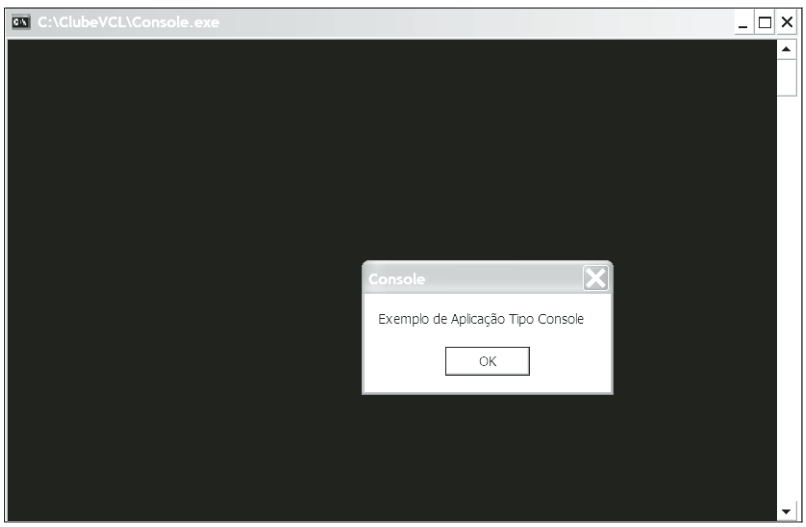


Figura 19.3: Acessando uma função da API do Windows em uma aplicação tipo console criada com o Delphi 7.

O objetivo deste tópico foi mostrar que o ambiente de desenvolvimento do Delphi 7 pode ser usado no desenvolvimento de aplicações do tipo console, utilizando-se o Pascal como linguagem de programação procedural.

Desta maneira, se você possui qualquer aplicação desenvolvida utilizando-se o Pascal tradicional, pode recompilá-la a partir do ambiente do Delphi 7, executando os procedimentos descritos neste tópico (talvez tenha alguma dificuldade no caso de utilização de algumas bibliotecas gráficas). Dessa maneira não é necessário manter instalados simultaneamente, em um mesmo equipamento, o Delphi 7 e uma versão antiga do compilador Turbo Pascal.

Nos próximos tópicos, a comparação entre a programação estruturada e a programação orientada a objetos será feita, inicialmente, utilizando aplicações tipo console, e nestas situações este tópico poderá ser bastante útil.

KNOW-HOW em: Fundamentos da Programação Orientada a Objetos

PRÉ-REQUISITOS

- Conhecimento das técnicas de programação estruturada.
- Conhecimento da sintaxe básica da linguagem Pascal.
- Familiaridade com o ambiente de desenvolvimento integrado do Borland Delphi 7.

METODOLOGIA

- Apresentação dos conceitos fundamentais da programação orientada a objetos.
- Comparação dos procedimentos adotados na programação estruturada e na programação orientada a objetos.

TÉCNICA

- Criação de classes e objetos com o Borland Delphi 7.

A Filosofia da Programação Orientada a Objetos

Nos anos 80, as técnicas de programação estruturada se destacaram como uma nova estratégia para o desenvolvimento de programas, adotando a técnica de "dividir para conquistar" – na qual um problema era subdividido em problemas menores, denominados módulos, procedimentos, funções ou sub-rotinas (a terminologia variava principalmente em função da linguagem de programação utilizada) – cada uma destinada a realizar uma determinada tarefa.

O problema era, logo, resolvido utilizando-se um módulo principal, que recorria a estes módulos secundários sempre que fosse necessário executar uma determinada tarefa. Dessa maneira, o programa principal tinha como função o gerenciamento da solução global – tornando-o mais claro e necessitando de uma menor codificação.

A partir das técnicas de programação estruturada surgiram as bibliotecas de funções, que poderiam ser compartilhadas entre vários programas – e no caso do Pascal bastava que se fornecesse o arquivo compilado – protegendo-se desta maneira o código-fonte da biblioteca.

Esta filosofia de programação, no entanto, apresentava uma característica que ainda dificultava a abordagem do problema a ser resolvido de uma forma mais próxima do mundo real: o fato de que os dados e as funções que operavam sobre estes dados eram tratados como entidades inteiramente distintas.

Consideremos, apenas para relembrar estes conceitos, o código apresentado a seguir – escrito em Pascal Procedural – e destinado a calcular o imposto de renda (pessoa física).

Este é um exemplo bastante simples, mas nosso objetivo aqui não é ensinar programação procedural, mas comparar a sintaxe utilizada na sua implementação com a utilizada na solução do mesmo problema usando as técnicas de programação orientada a objetos a serem apresentadas posteriormente.

As fórmulas utilizadas para o cálculo do imposto não correspondem às utilizadas atualmente pela Receita Federal, tendo sido incluídas apenas a título de exemplificação – portanto não as incorpore diretamente no código-fonte de seus programas.

```
program ImpostoDeRenda;

{$APPTYPE CONSOLE}

type
    TContribuinte = record
     rendimento, imposto : double;
end;

var
   Contribuinte : TContribuinte;

procedure calcula_imposto(var um_contribuinte : TContribuinte);
begin
     with um_contribuinte do
     begin
       if (rendimento < 900.0) then imposto := 0.0;
       if (rendimento > 1800) then imposto := 0.275*rendimento-360.0;
       if (rendimento >= 900)and(rendimento <= 1800) then imposto := 0.15*rendimento -135.0;
     end;
end;

begin
     write('Digite o valor do rendimento:');
     readln(contribuinte. rendimento);
     calcula_imposto(contribuinte);
     writeln('imposto:',contribuinte.imposto);
     readln;
end.
```

Observe que no Delphi 7, a janela para edição de códigos (O Code Editor) apresenta à sua esquerda uma outra janela denominada Code Browser, como mostrado na figura a seguir, e que exibe os procedimentos, tipos, objetos e variáveis definidos na unit que está sendo editada (mesmo se tratando de uma aplicação tipo console).

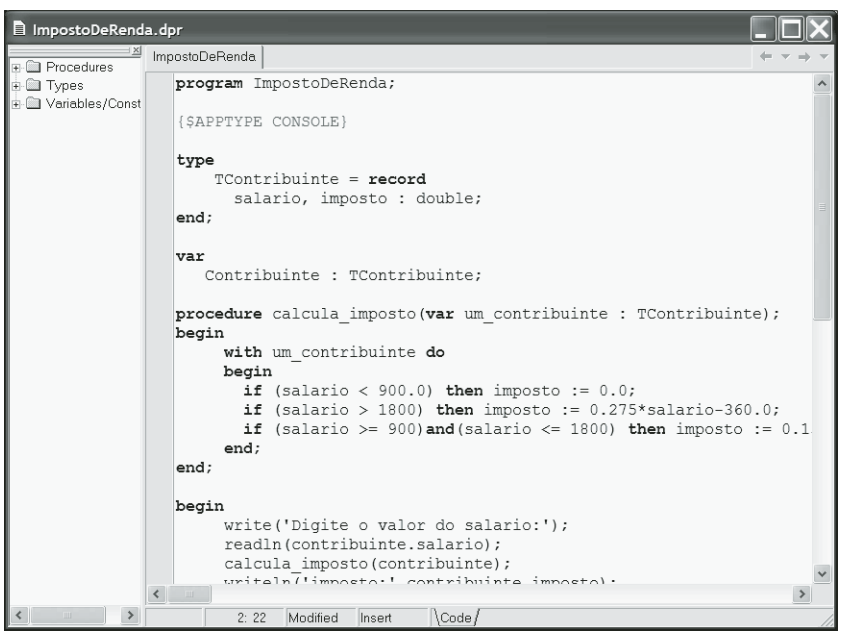


Figura 19.4: A janela do editor de códigos, com a janela do Code Browser incorporada.

Esta janela pode ser desacoplada do editor de códigos e visualizada de forma independente, bastando para isso selecionar a sua barra de títulos e arrastar a janela para outra região do ambiente de desenvolvimento.

No próximo tópico será apresentada uma análise deste código, escrito por meio do Pascal procedural.

Análise do Código-fonte

Conforme descrito anteriormente, nesse programa (escrito em Pascal Procedural) os dados são entidades totalmente distintas das funções.

Nesse caso particular, os dados são declarados no seguinte trecho de código:

```
var
   Contribuinte : TContribuinte;
```

onde TContribuinte é um tipo composto (registro) – definido na seção type da unit – que reúne duas variáveis chamadas rendimento e imposto (denominadas campos). Os tipos compostos já existem há muito tempo na linguagem pascal, e não são uma inovação do Delphi 7.

A única sub-rotina deste programa é a função calcula_imposto, que recebe como parâmetro uma variável do tipo TContribuinte (sub-rotina esta reproduzida a seguir).

```
procedure calcula_imposto(var um_contribuinte : TContribuinte);
begin
   with um_contribuinte do
   begin
       if (rendimento < 900.0) then imposto := 0.0;
       if (rendimento > 1800) then imposto := 0.275*rendimento - 360.0;
       if (rendimento >= 900)and(rendimento <= 1800) then imposto := 0.15*rendimento -135.0;
   end;
end;
```

Observe a utilização do parâmetro var, caracterizando que a variável é passada por referência, e não por valor. Desta maneira, garantimos que estamos trabalhando com a variável original, e não com uma cópia da mesma.

O programa principal é muito simples, e dispensa maiores comentários, pois realiza apenas a leitura das informações solicitadas, chama a função calcula_imposto e exibe o resultado final na tela.

Dessa maneira pode-se constatar que, conforme já dissemos anteriormente, na programação procedural os dados e as funções são entidades distintas.

É evidente que, neste exemplo muito simples, a variável Contribuinte possui escopo global, e por esta razão poderíamos alterá-la diretamente na sub-rotina, não havendo a necessidade de utilização do parâmetro especificado. Esta, no entanto, é uma prática de programação desaconselhável, principalmente se levarmos em conta a possibilidade de se incluir a procedure calcula_imposto em uma biblioteca de funções codificada em uma unit independente.

O objetivo da programação orientada a objetos, por outro lado, consiste primordialmente em reunir os dados, as funções e procedimentos que operam sobre estes dados em uma única entidade ou tipo,

denominada classe. Uma classe, portanto, será um novo tipo da linguagem na qual estarão reunidos dados e funções (ou procedimentos).

A Implementação de uma Classe

Na linguagem Object Pascal, a definição de uma classe é feita em um trecho de código que obedece à seguinte sintaxe:

```
type
     Nome_da_classe = Class
        Nome_do_dado_1 : tipo_1;
                ..........
        Nome_do_dado_n : tipo_n;
        Cabeçalho_da_função_1;
                ..........
        Cabeçalho _da_função_n;
     end;
```

onde "tipo" pode ser qualquer tipo básico definido pela linguagem, criado pelo programador ou, até mesmo, outras classes previamente definidas.

Repare que agora estamos definindo um novo tipo, no qual internamente são declaradas variáveis e funções. Pode-se dizer, sem muito preciosismo, que a definição de uma classe estende a definição de um registro, permitindo a incorporação de funções ao tipo que está sendo criado.

Por convenção, os nomes das classes definidas na linguagem Object Pascal começam com a letra "T", e neste livro seguiremos esta convenção.

É importante caracterizar que, na definição do tipo, as funções e os procedimentos são declarados mas não implementados. A implementação das funções e dos procedimentos declarados internamente a uma classe, e que no jargão da programação orientada a objetos são denominados métodos, é feita na seção Implementation da unit em que a classe é definida, devendo, no entanto, o nome do método ser precedido pelo nome da classe e obedecer à seguinte sintaxe:

```
function Nome_da_Classe.Nome_do_Método(parâmetros)
begin
     // Código do método
end;
```

ou

```
procedure Nome_da_Classe.Nome_do_Método(parâmetros)
begin
     // Código do método
end;
```

É evidente que um método, assim como uma função ou procedimento, também pode não ter parâmetros.

No caso do exemplo anterior, poderíamos, portanto, criar uma classe denominada TContribuinte, como mostrado a seguir.

```
type
    TContribuinte = Class
```

```
      rendimento, imposto : double;
      procedure calcula_imposto;
    end;
```

A implementação do método calcula_imposto, por sua vez, poderia ser feita da seguinte maneira:

```
procedure TContribuinte.calcula_imposto;
begin
    if (rendimento < 900.0) then imposto := 0.0;
    if (rendimento > 1800) then imposto := 0.275*rendimento -360.0;
    if (rendimento >= 900)and(rendimento <= 1800) then imposto := 0.15*rendimento -135.0;
end;
```

Repare que, neste caso, não há a necessidade de se passar qualquer parâmetro. Isso se deve ao fato de que, por ser definido internamente a uma classe, um método pode acessar diretamente todos os seus campos. Conseqüentemente, o método calcula_imposto acessa diretamente o campo rendimento, e este não precisa ser passado como parâmetro.

O Delphi 7 possui um recurso que permite a geração automática do código básico de um método na seção Implementation da unit, bastando para isso que se pressione, simultaneamente as teclas Ctrl, Shift e C. O mesmo se aplica à definição de propriedades para uma classe, conforme será visto posteriormente. Esse recurso, denominado Complementação Automática de Classes, acelera muito o trabalho de codificação por parte do programador, reduzindo-se também a possibilidade de erros de digitação.

Antes de utilizarmos uma variável, objeto ou instância de uma classe, devemos alocar memória para o mesmo.

A alocação de memória para um objeto é feita executando-se um método especial da classe, denominado método construtor. Em geral (mas não obrigatoriamente) esse método é denominado Create, e pode ou não possuir parâmetros. Você pode definir vários métodos construtores para uma classe, com nomes e parâmetros distintos, e decidir qual deve ser usado na alocação de memória para um objeto ou instância da classe.

Essa memória que foi alocada deve ser posteriormente liberada, antes de se finalizar a execução do programa, e isso é feito executando-se um outro método especial da classe, denominado método destrutor. Em geral, esse método é denominado Destroy, e pode ou não possuir parâmetros.

O programa anterior poderia, portanto, ser recodificado da seguinte maneira (usando uma classe ao invés de um registro):

```
program imposto_renda;

{$APPTYPE CONSOLE}

type
    TContribuinte = Class
      rendimento, imposto : double;
      procedure calcula_imposto;
    end;

var
   Contribuinte : TContribuinte;

procedure TContribuinte.calcula_imposto;
begin
```

```
    if (rendimento < 900.0) then imposto := 0.0;
    if (> 1800) then imposto := 0.275*rendimento -360.0;
    if (rendimento >= 900)and(rendimento <= 1800) then imposto := 0.15*rendimento -135.0;
end;

begin
     Contribuinte := TContribuinte.Create;
     write('Digite o valor do rendimento:');
     readln(contribuinte. rendimento);
     contribuinte.calcula_imposto;
     writeln('imposto:',contribuinte.imposto);
     Contribuinte.Destroy;
     readln;
end.
```

ANÁLISE DO CÓDIGO-FONTE

Nesse exemplo, a variável Contribuinte é, na realidade, um objeto ou instância (terminologia largamente empregada na programação orientada a objetos) da classe TContribuinte, mas não é nenhum pecado chamá-la de variável (embora os puristas da programação orientada a objetos possam querer me crucificar por causa disso).

Repare que, no trecho principal do programa, alterou-se a linha de código em que o procedimento para cálculo do imposto é executado.

No programa escrito anteriormente, utilizando as técnicas de programação estruturada, o procedimento para cálculo do imposto era uma entidade independente dos dados que deve manipular, e precisava receber, como parâmetro, uma variável do tipo composto TContribuinte.

O programa escrito utilizando as técnicas de programação orientada a objetos, por sua vez, define esse procedimento como um método da classe TContribuinte. Dessa maneira, a chamada desse procedimento (que nesse caso não possui parâmetros) é feita a partir de um objeto dessa classe, como mostra a linha de código a seguir:

```
contribuinte.calcula_imposto;
```

Além disso, foram incluídas duas outras linhas de código, em que os métodos construtor e destrutor da classe são executados. Repare que o método construtor é executado a partir da classe propriamente dita, enquanto o destrutor é executado a partir do objeto ou instância da classe.

Nesse ponto, uma dúvida deve estar presente na mente do prezado leitor: como os métodos construtor e destrutor podem ser executados, se não foram sequer definidos para a classe? A resposta a essa pergunta será dada no próximo tópico, em que abordaremos uma das características mais importantes da programação orientada a objetos – o mecanismo de herança de classes.

KNOW-HOW EM: HERANÇA DE CLASSES

PRÉ-REQUISITOS

- Conhecimento da sintaxe básica da linguagem Pascal.
- Familiaridade com o ambiente de desenvolvimento integrado do Delphi 7.

METODOLOGIA

- Apresentação do conceito de herança da programação orientada a objetos.

TÉCNICA

- Implementação do mecanismo de herança de classes com o Delphi 7.

O Conceito de Herança de Classes

Uma das principais vantagens da programação orientada a objetos, e muito divulgada na literatura especializada sobre o assunto, reside na reutilização de código através do mecanismo de herança.

Esse mecanismo permite que se definam novas classes (chamadas classes derivadas) a partir de uma classe fundamental (denominada classe-base).

Esse tipo de abordagem permite que a implementação da nossa solução para um determinado problema seja feita de uma maneira muito mais apropriada, e semelhante ao problema real.

Considere novamente o problema de elaboração de um programa para o cálculo do imposto de renda. No mundo real, existem dois tipos de contribuinte – pessoa física e pessoa jurídica. Esses dois tipos, no entanto, dispõem de algumas características em comum: ambos têm imposto a pagar, e devem ter um método para o cálculo desse imposto.

Podemos então imaginar uma classe-base, denominada classe TContribuinte, a partir da qual serão derivadas por herança duas outras classes, denominadas TPessoa_Fisica e TPessoa_Juridica.

Conseqüentemente, podemos definir, para a classe-base TContribuinte, campos chamados rendimento e imposto, do tipo double, e um método chamado calcula_imposto.

O esqueleto dessa classe é apresentado a seguir.

```
type
    TContribuinte = Class
      rendimento, imposto : double;
      procedure calcula_imposto;
    end;
```

A definição de uma classe derivada a partir de uma classe-base é feita mediante a utilização de um código com a seguinte sintaxe:

```
 Nome_Classe_Derivada = Class(Nome_Classe_Base)
        // Definição dos campos e métodos da classe derivada
 end;
```

Caso não seja especificado o nome de uma classe-base, será usada como tal a classe TObject, que é a classe-base de todas as classes definidas na linguagem Object Pascal. Se você analisar o diagrama de hierarquia de objetos do Delphi 7, verá que a classe TObject é exibida isoladamente no topo dessa hierarquia.

Dessa maneira, a classe TContribuinte poderia ser redefinida da seguinte maneira, em que o nome da classe-base (TObject) é exibido de maneira explícita.

```
type
    TContribuinte = Class(TObject)
```

```
      rendimento, imposto : double;
      procedure calcula_imposto;
    end;
```

Na unit System (definida pelo arquivo System.pas), a classe TObject é definida da seguinte maneira:

```
TObject = class
    constructor Create;
    procedure Free;
    class function InitInstance(Instance: Pointer): TObject;
    procedure CleanupInstance;
    function ClassType: TClass;
    class function ClassName: ShortString;
    class function ClassNameIs(const Name: string): Boolean;
    class function ClassParent: TClass;
    class function ClassInfo: Pointer;
    class function InstanceSize: Longint;
    class function InheritsFrom(AClass: TClass): Boolean;
    class function MethodAddress(const Name: ShortString): Pointer;
    class function MethodName(Address: Pointer): ShortString;
    function FieldAddress(const Name: ShortString): Pointer;
    function GetInterface(const IID: TGUID; out Obj): Boolean;
    class function GetInterfaceEntry(const IID: TGUID): PInterfaceEntry;
    class function GetInterfaceTable: PInterfaceTable;
    function SafeCallException(ExceptObject: TObject;
      ExceptAddr: Pointer): HResult; virtual;
    procedure AfterConstruction; virtual;
    procedure BeforeDestruction; virtual;
    procedure Dispatch(var Message); virtual;
    procedure DefaultHandler(var Message); virtual;
    class function NewInstance: TObject; virtual;
    procedure FreeInstance; virtual;
    destructor Destroy; virtual;
  end;
```

Observe que essa classe já define os métodos Create e Destroy. Conseqüentemente, as classes dela derivadas por herança (como, por exemplo, a classe TContribuinte) herdarão esses métodos da classe TObject. Isso esclarece a questão colocada no tópico anterior, em que se utilizaram os métodos Create e Destroy sem que os mesmos houvessem sido explicitamente declarados na classe. Esses métodos foram definidos na classe-base (TObject), podendo ser utilizados nas classes dela derivadas por herança.

Repare ainda que alguns métodos se iniciam com palavras reservadas especiais, como:

- Constructor, no caso do método construtor da classe.
- Destructor, no caso do método destrutor da classe.
- Class, que ocorre em diversos métodos.

O significado de cada uma dessas palavras reservadas será descrito nos próximos tópicos. Não comentaremos o significado das palavras reservadas procedure e function, por considerarmos o seu conhecimento obrigatório a todos aqueles que já têm alguma experiência com a linguagem pascal (incluindo-se neste grupo aqueles que já leram a primeira parte deste livro).

Dessa maneira, a definição das classes TContribuinte, TPessoa_Juridica e TPessoa_Fisica pode ser feita da seguinte maneira:

```
TContribuinte = Class(TObject)
      imposto : double;
```

```
      rendimento: double;
      procedure calcula_imposto;
    end;

TPessoa_Juridica = Class(TContribuinte)
end;

TPessoa_Fisica = Class(TContribuinte)
end;
```

Essas classes, portanto, herdam os campos imposto e o método calcula_imposto da classe-base TContribuinte. Além disso, herdam também os métodos definidos na classe TObject.

No jargão da programação orientada a objetos diz-se que as classes TPessoa_Fisica e TPessoa_Juridica são especializações da classe TContribuinte.

Quando você declarar uma instância de uma classe, aquela armazenará, na realidade, o endereço de um objeto dessa classe, podendo no entanto armazenar também o endereço de uma classe dela derivada por herança. Caso nenhum endereço seja referenciado, o seu valor será igual a nil (nulo).

Consideremos, por exemplo, a linha de código a seguir, em que declaramos uma instância da classe TContribuinte:

```
Contribuinte : TContribuinte;
```

Essa instância – a variável contribuinte – pode armazenar o endereço de um objeto da classe TContribuinte ou de qualquer das classes dela derivadas por herança.

Dessa maneira, as linhas de código a seguir são igualmente válidas:

```
Contribuinte := TContribuinte.Create;
Contribuinte := TPessoa_Fisica.Create;
Contribuinte := TPessoa_Juridica.Create;
```

A recíproca, no entanto, não é verdadeira: você não pode declarar um objeto de uma classe e referenciar um objeto da sua classe-base.

Métodos Construtores

Um construtor é um método especial de uma classe, que tem por finalidade alocar a memória para um objeto que está sendo criado (ou instanciado, no jargão da Programação Orientada a Objetos) e, opcionalmente, atribuir valores iniciais aos campos do objeto.

Uma classe pode ter mais do que um método construtor e, se você não definir explicitamente um construtor para essa classe, o construtor da sua classe-base será utilizado.

A definição de um construtor utiliza a palavra-chave constructor em vez de procedure ou function. Além disso, um construtor pode ter parâmetros e chamar o construtor da classe-base. Dessa maneira, é a palavra-chave constructor que indica ao compilador que deve alocar a memória necessária para o objeto.

Embora não especifique um tipo de retorno, um construtor retorna um novo objeto da classe. A chamada ao método construtor deve ser feita a partir da definição da classe, e não a partir de uma instância ou

objeto da classe. Por essa razão, nos tópicos anteriores, a chamada do método Create foi feita a partir da classe TContribuinte, e não a partir do objeto Contribuinte.

Alguns autores costumam identificar uma classe como uma "fábrica de objetos". Nesse caso, o método construtor seria uma "máquina" existente na fábrica, e que retorna um objeto da classe sempre que posta em funcionamento.

Caso você execute um construtor a partir de uma instância da classe, nenhuma memória será alocada e não será retornado um novo objeto da classe – apenas o código definido no corpo do construtor será executado (esta, no entanto, não é uma prática aconselhável – você deve chamar o construtor a partir da definição da classe – e não a partir de uma instância ou objeto).

Métodos Destrutores

Um destrutor é um método especial de uma classe, que tem por finalidade liberar a memória alocada para um objeto ou instância da classe.

Uma classe pode ter mais do que um método destrutor e, se você não definir explicitamente um destrutor para essa classe, o destrutor da sua classe-base será utilizado.

A definição de um método destrutor usa a palavra-chave destructor em vez de procedure ou function. Além disso, um método destrutor pode ter parâmetros e chamar o método destrutor da classe-base.

Se você está criando uma nova classe derivada de uma classe-base e está sobrecarregando ou redefinindo o seu método destrutor, a última instrução a ser colocada no corpo do novo método destrutor deve ser aquela que chama o destrutor da classe-base, principalmente se o destrutor da classe-base executa algumas tarefas importantes relacionadas à liberação de memória alocada para um objeto. No caso de uma classe derivada de TObject, no entanto, isso não é necessário, pois seu método destrutor não realiza nenhuma tarefa importante, como mostra o trecho de código a seguir (extraído da unit System.pas):

```
destructor TObject.Destroy;
begin
end;
```

Geralmente, é mais seguro chamar o método Free do que o método Destroy de uma classe, uma vez que Free verifica se o objeto realmente existe na memória e, em caso positivo, executa o método Destroy. Esse método também é declarado como uma procedure na classe TObject, e sua implementação (definida na unit System.pas da VCL do Delphi 7) é reproduzida a seguir.

```
procedure TObject.Free;
asm
        TEST    EAX,EAX
        JE      @@exit
        MOV     ECX,[EAX]
        MOV     DL,1
        CALL    dword ptr [ECX].vmtDestroy
@@exit:
end;
```

Esse código, na realidade, está escrito em Assembler, e verifica se existe memória alocada para o objeto antes de executar o seu destrutor.

Você não deve se preocupar em compreender o significado desse código, afinal de contas você não está lendo um texto sobre Assembler. O importante é compreender que, mesmo que não exista memória alocada para o objeto, a execução desse método não provocará um erro em run-time.

Para compreender a diferença existente entre a utilização dos métodos Free e Destroy, crie uma aplicação bastante simples, composta por um único formulário, no qual são inseridos uma caixa de texto (chamada Edit1) e um botão de comando. No procedimento associado ao evento OnClick do botão de comando digite a seguinte linha de código:

```
Edit1.Free;
```

Execute a aplicação e selecione o botão. A caixa de texto será eliminada do formulário, e se você selecionar novamente o botão de comando nada de anormal acontecerá. Por outro lado, se você substituir a linha de código anterior pela apresentada a seguir, o componente caixa de texto será eliminado do formulário quando o botão de comando for selecionado pela primeira vez. Entretanto, caso você selecione novamente o botão de comando, a execução da aplicação será interrompida e uma mensagem de erro será exibida. Isso se deve ao fato de que o método Destroy, ao contrário do método Free, não verifica se realmente existe memória alocada para o objeto antes de tentar destruí-lo.

```
Edit1.Destroy;
```

Mas atenção: o método Free verifica se existe memória alocada para o objeto, isto é, se o valor referenciado pelo objeto é diferente de nil antes de executar o método Destroy para destruir o objeto. Contudo, cabe a você garantir que, após destruir o objeto, esse apontará para um endereço nulo, isto é, nil. Para garantir que isso ocorra, no entanto, você deve atribuir o valor nil à instância após executar o seu método Free, no caso de estar implementando esses métodos da classe.

Visibilidade dos Campos e Métodos de uma Classe

Define-se como visibilidade dos campos e métodos de uma classe o conjunto de trechos de código em que os mesmos podem ser acessados.

Na linguagem Object Pascal, os métodos e campos de uma classe podem ser classificados como públicos (public), privados (private) ou protegidos (protected). No caso de classes que representam componentes, temos ainda o tipo published (não traduzirei esse termo), que será abordado no capítulo relativo à criação de componentes. As classes derivadas da classe TAutoObject (usadas em automação OLE) também poderão ter campos do tipo automated.

Por enquanto, vamos nos concentrar apenas nos três primeiros tipos: public, private e protected.

Campos e Métodos Públicos (public)

Os campos e métodos públicos de uma classe são aqueles que podem ser acessados em qualquer trecho de um programa no qual um objeto ou instância da classe é definido.

No Delphi 7 os campos de uma classe são public por default, isto é, caso não haja qualquer declaração explícita em relação a um campo de uma classe, este será considerado como um campo público.

Dessa maneira, as duas declarações a seguir são equivalentes:

```
type
 T Contribuinte = Class
           imposto : double;
           rendimento: double;
           procedure calcula_imposto;
    end;

type
TContribuinte = Class
   public
         imposto : double;
         rendimento: double;
         procedure calcula_imposto;
end;
```

Campos e Métodos Privados (private)

Os campos e métodos privados de uma classe são aqueles que só podem ser acessados por outros métodos da classe, isto é, só são visíveis dentro da classe, não podendo ser acessados nem mesmo nas classes dela derivadas por herança. Na declaração da classe, os métodos e campos privados devem ser precedidos da palavra-chave private, como mostrado no trecho de código a seguir.

```
type

TContribuinte = Class
   public
      imposto : double;
      rendimento: double;
      procedure calcula_imposto;
   private
      // Campos e métodos privados devem ser definidos aqui.
end;
```

Existe, no entanto, uma exceção a essa regra: uma classe pode acessar os campos e métodos privados de outras classes que tenham sido definidas na mesma unit.

Campos e Métodos Protegidos (protected)

Os campos e métodos protegidos de uma classe são aqueles que só podem ser acessados por outros métodos da classe ou de classes dela derivadas – isto é, só são visíveis dentro da própria classe ou das classes que forem dela derivadas por herança. Na declaração da classe, os métodos e campos protegidos devem ser precedidos da palavra-chave protected, como mostrado no trecho de código a seguir.

```
type
TContribuinte = Class
   public
      imposto : double;
      rendimento: double;
      procedure calcula_imposto;
   private
      // Campos e métodos privados devem ser definidos aqui.
   protected
      // Campos e métodos protegidos devem ser definidos aqui.
end;
```

Sobreposição de Métodos

Existem situações em que, ao criarmos uma classe, desejamos sobrepor o método existente na classe-base. Quando desejamos simplesmente substituir o método já existente na classe-base basta que se defina, na classe derivada, um método com o mesmo nome do método a ser sobreposto.

Desse modo, a classe derivada passará a ignorar a existência de um método de mesmo nome na classe-base.

Nada impede, no entanto, que você ainda acesse o método de mesmo nome da classe-base, desde que o nome do método seja precedido pela palavra reservada inherited.

Para compreender esse mecanismo, codifique uma aplicação tipo console da maneira indicada a seguir.

```
program Exemplo;

{$APPTYPE CONSOLE}

uses
  Dialogs;

type
  TClasseBaseTeste = class
      procedure ExibeMensagem;
  end;

  TClasseDerivadaTeste = class(TClasseBaseTeste)
      procedure ExibeMensagem;
  end;

procedure TClasseBaseTeste.ExibeMensagem;
begin
    ShowMessage('Classe Base');
end;

procedure TClasseDerivadaTeste.ExibeMensagem;
begin
    ShowMessage('Classe Derivada');
    inherited ExibeMensagem;
end;

var
   Teste : TClasseDerivadaTeste;

{$R *.RES}

begin
   Teste := TClasseDerivadaTeste.Create;
   writeln('Digite Qualquer tecla');
   Readln;
   Teste.ExibeMensagem;
   Teste.Free;
end.
```

Execute essa aplicação e verifique que, embora o método ExibeMensagem da classe-base tenha sido sobreposto na classe derivada, pode ainda ser chamado em um método da classe derivada usando a palavra reservada inherited. Experimente trocar a ordem das linhas de código do método ExibeMensagem da classe derivada, e verifique que a ordem de exibição de mensagens é alterada.

Métodos Estáticos, Virtuais e Dinâmicos

Os métodos considerados até o momento são denominados estáticos, que são aqueles cujo endereço é determinado pelo compilador durante a compilação do programa e geração do arquivo executável.

Existem situações, no entanto, em que não sabemos de antemão que tipo de objeto será apontado por uma determinada variável (lembre-se de que uma variável de uma determinada classe pode armazenar um objeto dessa classe ou de qualquer classe dela derivada por herança).

Nesse caso, para que o programa possa definir durante a sua execução que método deve ser chamado, devemos declarar o método como virtual.

Consideremos novamente uma aplicação console bastante simples, com o código apresentado a seguir.

```
program Exemplo;

{$APPTYPE CONSOLE}

uses
  Dialogs;

type
  TClasseBaseTeste = class
      procedure ExibeMensagem;
  end;

  TClasseDerivadaTeste = class(TClasseBaseTeste)
      procedure ExibeMensagem;
  end;

procedure TClasseBaseTeste.ExibeMensagem;
begin
    ShowMessage('Classe Base');
end;

procedure TClasseDerivadaTeste.ExibeMensagem;
begin
    ShowMessage('Classe Derivada');
end;

var
   Teste : TClasseBaseTeste;

{$R *.RES}

begin
   Teste := TClasseDerivadaTeste.Create;
   writeln('Digite Qualquer tecla');
   Readln;
   Teste.ExibeMensagem;
   Teste.Free;
end.
```

Nesse caso, embora a variável Teste seja declarada como um objeto da classe TClasseBaseTeste, no programa principal referencia um objeto da classe TClasseDerivadaTeste – o que é perfeitamente legal, pois, conforme discutido anteriormente, uma instância ou objeto de uma determinada classe pode referenciar um objeto dessa classe ou de qualquer classe dela derivada por herança (embora a recíproca não seja correta).

Entretanto, ao executar esse código verifica-se que o método executado é o definido na classe-base, e não o definido na classe derivada. Isto se deve à utilização de métodos estáticos, cujo endereço é determinado na fase de compilação – e nessa fase a variável Teste é declarada como um objeto da classe TClasseBaseTeste (embora durante a execução da aplicação ela venha a referenciar um objeto da classe TClasseDerivadaTeste). Conseqüentemente, o programa executará sempre o método da classe-base, pois durante a compilação definiu-se Teste como um objeto da classe TBaseTeste.

A solução para esse problema está justamente na utilização de métodos virtuais – cujo endereço é definido durante a execução do aplicativo (e não durante a sua compilação).

Para declarar um método como virtual, deve-se acrescentar a palavra-chave virtual, seguida de um ponto-e-vírgula, ao final da declaração do método na classe-base. Nas classes derivadas em que esse método for sobrecarregado, deve-se acrescentar a palavra-chave override (em vez de virtual), seguida de um ponto-e-vírgula, ao final da declaração do método.

Dessa maneira, o programa anterior poderia ser recodificado da seguinte maneira:

```
program Exemplo;

uses
  Dialogs;

type
  TClasseBaseTeste = class
      procedure ExibeMensagem;virtual;
  end;

  TClasseDerivadaTeste = class(TClasseBaseTeste)
      procedure ExibeMensagem;override;
  end;

procedure TClasseBaseTeste.ExibeMensagem;
begin
    ShowMessage('Classe Base');
end;

procedure TClasseDerivadaTeste.ExibeMensagem;
begin
    ShowMessage('Classe Derivada');
end;

var
   Teste : TClasseBaseTeste;

begin
   Teste := TClasseDerivadaTeste.Create;
   writeln('Digite Qualquer tecla');
   Readln;
   Teste.ExibeMensagem;
   Teste.Free;
end.
```

Se você executar essa aplicação, verá que o método correto é chamado. Nesse caso, no entanto, nada impede que o método da classe-base continue a ser acessado pelo método de mesmo nome da classe derivada usando-se a palavra reservada inherited – mas, nesse caso, o método correto será chamado!

Diferentemente do que ocorreu no exemplo anterior, ao encontrar a palavra-chave virtual na definição do método ExibeMensagem, o compilador ficou ciente de que o endereço do método a ser executado será determinado durante a execução do aplicativo, e não durante a sua compilação – e isso vai depender do tipo de classe realmente referenciado pelo objeto.

Mas e os Métodos Dinâmicos? O Que Significam?

A finalidade dos métodos dinâmicos é a mesma dos métodos virtuais, isto é, são funcionalmente equivalentes. A única diferença diz respeito à otimização feita pelo compilador. Ao utilizar métodos virtuais, melhora-se o desempenho em detrimento do tamanho do código gerado. Por outro lado, a utilização de métodos dinâmicos reduz o tamanho do código em detrimento do desempenho.

Para utilizar o recurso de métodos dinâmicos, basta recodificar o programa-exemplo como mostrado a seguir.

```
program Exemplo;

{$APPTYPE CONSOLE}

uses
  Dialogs;

type
  TClasseBaseTeste = class
      procedure ExibeMensagem;dynamic;
  end;
  TClasseDerivadaTeste = class(TClasseBaseTeste)
      procedure ExibeMensagem;override;
  end;

procedure TClasseBaseTeste.ExibeMensagem;
begin
    ShowMessage('Classe Base');
end;

procedure TClasseDerivadaTeste.ExibeMensagem;
begin
    ShowMessage('Classe Derivada');
end;

var
   Teste : TClasseBaseTeste;

{$R *.RES}

begin
   Teste := TClasseDerivadaTeste.Create;
   writeln('Digite Qualquer tecla');
   Readln;
   Teste.ExibeMensagem;
   Teste.Free;
end.
```

Métodos Abstratos

Os métodos abstratos formam um tipo especial de método, que são declarados em uma classe-base, mas só são implementados nas classes dela derivadas por herança.

Esse tipo de método é útil quando se quer definir um método que deverá ser implementado em todas as classes derivadas de uma classe-base.

Um método é declarado como abstrato ao se acrescentar a palavra-chave abstract ao final da sua declaração. Entretanto, um método só pode ser declarado como abstrato se é previamente declarado como virtual ou dinâmico. O trecho de código a seguir mostra a declaração de um método abstrato.

```
type
 Figura = class
   procedure Desenha; virtual; abstract;
  ...
 end;
```

Você pode sobrecarregar um método abstrato da mesma forma que sobrecarrega um método virtual ou dinâmico. A única restrição é que, nesse caso, não se pode usar a palavra reservada inherited para chamar o método de mesmo nome da classe-base (já que nesta ele não foi implementado).

Métodos de Classe

Conforme pode ser verificado na definição da classe TObject, a declaração de alguns de seus métodos começa com a palavra reservada class. Quando a declaração de um método inicia com essa palavra reservada, isso significa que esse método é intrínseco à definição da classe e não pode, no entanto, acessar os campos e métodos da classe.

Esses métodos serão descritos no próximo tópico, que discute em mais detalhes a classe TObject.

Propriedades

Nos tópicos anteriores foram apresentados os conceitos de campos e métodos de uma classe. Conforme já descrito anteriormente, os campos armazenam valores que definem as características de um objeto e os métodos implementam a sua funcionalidade.

Existem situações, no entanto, em que não se quer permitir o acesso direto aos campos de uma classe, ainda que esses sejam definidos como públicos. Nesse caso, uma opção natural seria a utilização de métodos destinados a ler e atribuir valores aos campos da classe.

Consideremos, por exemplo, a seguinte definição para a classe TContribuinte.

```
type
TContribuinte = Class
   public
      rendimento:double;
      imposto : double;
      procedure calcula_imposto;
   private
      // Campos e métodos privados devem ser definidos aqui.
   protected
      // Campos e métodos protegidos devem ser definidos aqui.
end;
```

Nesta definição, imposto é um campo público, e conseqüentemente pode ter o seu valor diretamente alterado, o que não caracteriza uma boa prática de programação, pois o valor do imposto deve sempre

ser calculado, e não ter um valor atribuído de forma direta (os puristas da orientação a objetos dizem que este campo deve ser encapsulado).

Uma solução natural, portanto, seria implementar o método calcula_imposto como uma função em vez de um procedimento, redefinindo a classe da maneira indicada a seguir.

```
TContribuinte = Class
   public
     rendimento:double;
     function calcula_imposto:double;
   private
      // Campos e métodos privados devem ser definidos aqui.
      imposto : double;
   protected
      // Campos e métodos protegidos devem ser definidos aqui.
end;
```

Nesse caso, o campo imposto passa a ser privado, e não público.

A sua implementação correspondente seria:

```
function TContribuinte.calcula_imposto:double;
begin
     if (rendimento < 900.0) then imposto := 0.0;
     if (rendimento > 1800) then imposto := 0.275*rendimento -360.0;
     if (rendimento >= 900)and(rendimento <= 1800) then imposto := 0.15*rendimento -135.0;
     result := imposto;
end;
```

Dessa maneira, se Contribuinte é um objeto da classe TContribuinte, a obtenção do valor do imposto pode ser feita utilizando-se a seguinte linha de código, onde valor_do_imposto é uma variável global do tipo double:

```
valor_do_imposto := Contribuinte.calcula_imposto;
```

O Delphi 7, no entanto, permite uma solução alternativa, que consiste em empregar o conceito de propriedades.

Uma propriedade consiste em um meio de comunicação entre um campo interno e privado da classe e o código que utiliza um objeto dessa classe.

Para o programador que utiliza a classe, uma propriedade é funcionalmente equivalente a um campo, mas para quem definiu a classe um campo e uma propriedade são entidades distintas.

Por convenção, os campos internos das classes definidas pela VCL e pela CLX (as bibliotecas de classes do Delphi 7) são privados e seus nomes começam com a letra F (de Field). As propriedades, por sua vez, têm o mesmo nome de um campo da classe, omitindo-se a letra F inicial.

O componente Label, por exemplo, é um objeto da classe TLabel, que tem uma propriedade chamada Left. Embora você, usuário da classe, acesse essa propriedade da mesma maneira que acessa um campo público qualquer, está na realidade acessando um meio de comunicação com um outro campo interno e privado da classe, chamado FLeft.

No caso da classe TContribuinte, por exemplo, poderíamos ter um campo privado chamado Fimposto e uma propriedade chamada imposto.

A definição de uma propriedade, no entanto, é feita mediante a inclusão de uma linha de código que apresenta a seguinte sintaxe:

```
property nome_propriedade : tipo read metodo_ leitura write metodo_ escrita;
```

Como pode ser visto na definição anterior, uma propriedade deve ter um tipo e pode ter um método de leitura e um método de escrita. Quando se omite o nome do método de escrita, a propriedade é do tipo read-only (apenas de leitura) e quando se omite o método de leitura a propriedade é do tipo write-only (apenas de escrita). Existem situações, no entanto, que, em vez de se definir um método de leitura e um método de escrita, coloca-se em seu lugar o nome do campo em que o valor da propriedade será armazenado. Nesses casos, diz-se que a propriedade acessa diretamente o valor de um campo, sem que se use qualquer método específico. Conforme será visto posteriormente, os métodos de acesso são úteis para validar o valor atribuído a uma propriedade.

A propriedade imposto poderia, portanto, ser definida da seguinte maneira:

```
property imposto : double read calcula_imposto;
```

Nesse caso, imposto é definida como uma propriedade apenas de leitura (repare que foi omitida a cláusula write).

```
A classe TContribuinte seria, portanto, redefinida da seguinte maneira:
TContribuinte = Class
   public
      rendimento:double;
      function calcula_imposto:double;
      property imposto : double read calcula_imposto;
   private
      // Campos e métodos privados devem ser definidos aqui.
      Fimposto : double;
   protected
      // Campos e métodos protegidos devem ser definidos aqui.
end;
function TContribuinte.calcula_imposto:double;
begin
     if (rendimento < 900.0) then Fimposto := 0.0;
     if (rendimento > 1800) then Fimposto := 0.275*rendimento -360.0;
     if (rendimento >= 900)and(rendimento <= 1800) then Fimposto := 0.15*rendimento -135.0;
     result := Fimposto;
end;
```

No próximo tópico apresentaremos a classe TObject, que é a mãe de todas as classes definidas na linguagem Object Pascal.

Referência: a Classe TObject

Conforme descrito nos tópicos anteriores, a classe TObject é a classe-base de todas as classes implementadas no Delphi 7, isto é, toda classe é derivada por herança, ainda que indiretamente, da classe TObject.

É importante, portanto, que você como desenvolvedor conheça detalhadamente as características dessa classe, conforme será mostrado nos próximos tópicos.

Principais Métodos da Classe TObject

Apresentamos a seguir uma descrição dos principais métodos definidos na classe TObject. Como a maioria desses métodos está implementada em Assembler, não discutiremos sua implementação – apenas descrevemos a finalidade de cada um deles.

AfterConstruction

Declaração

```
procedure AfterDestruction; virtual;
```

Esse método é um método virtual, executado automaticamente após a finalização do último método construtor da classe.

BeforeDestruction

Declaração

```
procedure BeforeDestruction; virtual;
```

Esse método é um método virtual, executado automaticamente antes da inicialização do primeiro método destrutor da classe.

Create

Declaração

```
constructor Create;
```

Esse método é o método construtor da classe, conforme indicado pela palavra reservada constructor, e tem por finalidade alocar a memória necessária a um objeto ou instância da classe. Como qualquer construtor, é executado como se fosse um método da classe (embora, por ser um método construtor, não seja precedido pela palavra reservada class).

ClassInfo

Declaração

```
class function ClassInfo: Pointer;
```

Como pode ser verificado a partir da sua declaração, este é um método de classe e, por ser uma função, deve retornar um valor, que nesse caso será um Pointer (ponteiro para um endereço de memória sem uma qualificação de tipo).

A Unit TypInfo define, no entanto, alguns tipos compostos que simplificam a utilização da informação retornada por esse método.

Dentre esses tipos compostos destaca-se:

```
TTypeInfo = record
    Kind: TTypeKind;
    Name: ShortString;
   {TypeData: TTypeData}
  end;
PtypeInfo: ^TtypeInfo.
```

O tipo TTypeInfo possui os campos Kind e Name, que retornam respectivamente um valor que indica o tipo de dado armazenado em um endereço e o nome do tipo ou Classe do valor armazenado nesse endereço, respectivamente.

- O tipo PtypeInfo representa um ponteiro para um valor do tipo TtypeInfo.
- O tipo TTypeKind (definido como um tipo enumerado em Pascal) também é declarado na unit TypeInfo, sendo sua definição reproduzida a seguir.

```
TTypeKind = (tkUnknown, tkInteger, tkChar, tkEnumeration, tkFloat,
    tkString, tkSet, tkClass, tkMethod, tkWChar, tkLString, tkWString,
    tkVariant, tkArray, tkRecord, tkInterface);
```

Logo, para obter informações sobre uma classe através do método ClassInfo, você precisa:

- Declarar uma variável do tipo PtypeInfo (que será um ponteiro).
- Armazenar nessa variável o valor retornado pelo método ClassInfo.
- Utilizar as informações armazenadas nos campos dessa variável.

ClassName

Declaração

```
class function ClassName: ShortString;
```

Como pode ser verificado a partir da sua declaração, esse é um método de classe e, por ser uma função, deve retornar um valor, que nesse caso será uma string que armazenará o nome da classe.

ClassNameIs

Declaração

```
class function ClassNameIs(const Name: string): Boolean;
```

Como pode ser verificado a partir da sua declaração, esse é um método de classe e, por ser uma função, deve retornar um valor – que nesse caso será true ou false – dependendo do texto definido na string passada como parâmetro.

Se o texto passado como parâmetro for igual ao nome da classe, esse método retornará true. Caso contrário, retornará o valor False.

ClassParent

Declaração

```
class function ClassParent: TClass;
```

Como pode ser verificado a partir da sua declaração, esse é um método de classe e, por ser uma função, deve retornar um valor – que nesse caso será uma referência à sua classe-base.

Observe que o valor retornado por esse método é do tipo TClass, declarado da seguinte maneira na unit System.pas:

```
TClass = class of TObject;
```

Isso significa que TClass é, na realidade, uma variável que pode armazenar o nome de um tipo de classe, que pode ser a classe TObject, ou qualquer classe dela derivada por herança.

Embora esse método não seja útil para a classe TObject (que não possui classe-base) pode ser empregada nas demais classes dela derivadas por herança. No caso de TObject, esse método retorna nil.

ClassType

Declaração

```
function ClassType: TClass;
```

Como pode ser verificado pela sua declaração, esse método retorna a classe do objeto, e é usado internamente pelo Delphi 7 no emprego dos operadores Is e As, a serem descritos posteriormente.

CleanupInstance

Declaração

```
procedure CleanupInstance;
```

Esse método é automaticamente executado a partir do método Free, e não deve ser chamado diretamente pelo código da sua aplicação. Esse método atribui uma string vazia a campos do tipo string e o valor Unassigned a campos do tipo Variant da classe.

Dispatch

Declaração

```
procedure Dispatch(var Message);
```

Esse método é responsável por disparar a execução de outros métodos da classe que sejam capazes de manipular mensagens do Windows, baseado no valor passado pelo parâmetro Message.

Caso nenhum dos métodos definidos pela classe (ou pelas suas classes ancestrais) seja capaz de manipular a mensagem indicada pelo parâmetro Message, o método DefaultHandler da classe será executado.

Isso permite que se dê um tratamento às mensagens que não são manipuladas diretamente pelos objetos da classe (o conceito de mensagens será apresentado posteriormente).

DefaultHandler

Declaração

```
procedure DefaultHandler(var Message); virtual;
```

Como pode ser verificado, esse é um método virtual, e que pode ser sobrecarregado nas classes derivadas por herança.

Conforme descrito no tópico anterior, esse método é automaticamente chamado pelo método Dispatch, sempre que a mensagem não for diretamente manipulada por um dos métodos da classe.

Destroy

Declaração

```
destructor Destroy; virtual;
```

Esse método é o método destrutor da classe, conforme indicado pela palavra reservada destructor, e tem por finalidade liberar a memória reservada para um objeto (memória esta alocada pelo seu construtor).

Como pode ser verificado, esse é um método virtual e que pode ser sobrecarregado nas classes derivadas.

Conforme descrito anteriormente, geralmente é mais seguro executar o método Free, que verifica se o valor armazenado na instância da classe é igual a nil, e só executa o método Destroy se existir memória alocada para o objeto.

FieldAddress

Declaração

```
function FieldAddress(const Name: ShortString): Pointer;
```

Esse método retorna, na forma de um Pointer (ponteiro para um endereço de memória sem uma qualificação de tipo), o endereço de um campo do objeto da classe, cujo nome é passado como parâmetro na forma de uma string. Caso não exista o campo cujo nome é passado como parâmetro, o método retornará nil.

Free

Declaração

```
procedure Free;
```

Esse método, conforme já descrito anteriormente, verifica se existe memória alocada para uma instância de uma classe, antes de chamar o método destrutor da classe.

Lembre-se de que cabe a você incluir, na implementação do código da sua aplicação, uma linha de código que atribua o valor nil à instância da classe após a execução do método Free (isso não é necessário para objetos da VCL e da CLX).

FreeInstance

Declaração

```
procedure FreeInstance; virtual;
```

Esse método libera a memória alocada previamente para uma instância de uma classe através de uma chamada ao seu método NewInstance (a ser descrito a seguir).

Esse método não precisa ser manipulado diretamente pelo código da sua aplicação. Entretanto, caso você tenha sobrecarregado o método NewInstance da classe, deve também sobrecarregar esse método, codificando as alterações necessárias à manutenção da compatibilidade entre esses métodos.

NewInstance

Declaração

```
class function NewInstance: TObject; virtual;
```

Esse método é um método de classe, e é responsável por alocar a memória necessária a uma nova instância da classe, retornando o endereço de memória a partir do qual estará armazenada esta instância recém-criada.

Para obter a quantidade de memória a ser alocada para a nova instância, esse método chama o método InstanceSize da classe, e para criar a nova instância executa o método InitInstance.

Esse método é chamado automaticamente pelo construtor da classe, e o código da sua aplicação não deve incluir uma chamada explícita a esse método.

InitInstance

Declaração

```
class procedure InitInstance(Instance: Pointer): TObject;
```

Esse método é um método de classe, e é responsável por criar a instância da classe propriamente dita (a ser retornada pelo construtor).

Você não precisa incluir uma chamada explícita a esse método no código da sua aplicação, pois isso é feito pelo método NewInstance. Entretanto, caso você sobrecarregue o método NewInstance, deve incluir, como último comando a ser executado por esse método, uma chamada ao método InitInstance.

InstanceSize

Declaração

```
class function InstanceSize: Longint;
```

Esse é um método de classe, e determina a quantidade de bytes de memória necessários para armazenar uma instância da classe.

Geralmente esse método é usado internamente pelos métodos responsáveis pela alocação de memória para um objeto, e posterior liberação dessa memória, quando o objeto for destruído.

Se você possuir um formulário com um componente chamado Edit1, da classe TEdit e um botão de comando chamado Button1 (da classe TButton) pode descobrir o número de bytes ocupados para cada

um desses componentes incluindo as seguintes linhas de código no procedimento associado ao evento OnClick desse botão de comando:

```
ShowMessage(IntToStr(Edit1.InstanceSize));
ShowMessage(IntToStr(Button1.InstanceSize));
```

Ao executar este procedimento, você descobrirá que Edit1 e Button1 ocupam 496 e 484 bytes, respectivamente.

MethodAddress

Declaração

```
class function MethodAddress(const Name: ShortString): Pointer;
```

Esse é um método de classe, e retorna, na forma de um Pointer (ponteiro para um endereço de memória sem uma qualificação de tipo), o endereço de um método cujo nome é passado como parâmetro na forma de uma string. Caso não exista o método cujo nome é passado como parâmetro, o método retornará nil.

Geralmente esse método é usado internamente pelo Delphi 7, e normalmente não deverá ser incluída uma chamada explícita a esse método no código da sua aplicação.

MethodName

Declaração

```
class function MethodName(Address: Pointer): ShortString;
```

Esse é um método de classe, e retorna, na forma de uma string, o nome do método cujo endereço é passado como parâmetro, na forma de um Pointer (ponteiro para um endereço de memória sem uma qualificação de tipo). Repare que esse método executa a tarefa inversa do método MethodAddress. Caso não exista o método cujo endereço é passado como parâmetro, o método retornará uma string vazia.

Geralmente esse método é usado internamente pelo Delphi 7, e normalmente não deverá ser incluída uma chamada explícita a esse método no código da sua aplicação.

InheritsFrom

Declaração

```
class function InheritsFrom(AClass: TClass): boolean;
```

Este é um método de classe, e determina se a classe cujo tipo é passado como parâmetro é ou não um ancestral da classe a partir da qual o método é executado

Evidentemente, se você passar TObject como parâmetro, esse método retornará sempre o valor true, pois todas as classes são derivadas, ainda que indiretamente, da classe TObject.

KNOW-HOW em: Conversão de Tipos

PRÉ-REQUISITOS

- Conhecimento da sintaxe básica da linguagem Pascal e dos fundamentos da programação orientada a objetos em Delphi 7 .
- Familiaridade com os ambientes de desenvolvimento integrado do Delphi 7.

METODOLOGIA

- Apresentação do conceito de conversão de tipos.

TÉCNICA

- Implementação do mecanismo de conversão de tipos, usando os operadores Is, As e a conversão explícita entre classes.

O Conceito de Conversão de Tipos

A conversão de tipos permite que se trate um objeto de uma classe como uma instância de uma das suas classes-base.

Essa técnica é muito útil nas seguintes situações:

- Quando se quer realizar uma mesma operação sobre objetos de classes distintas, mas que possuem uma característica semelhante, presente em uma classe-base comum.
- Quando se quer compartilhar um procedimento (que pode estar ou não associado a um evento) entre vários objetos.

Nesse caso, a solução consiste em tratar os vários objetos como se fossem uma instância da classe-base da qual todas são derivadas (direta ou indiretamente).

Para verificar se um objeto é uma instância de uma classe derivada de uma classe-base, deve-se utilizar o operador Is.

Para tratar um objeto como uma instância de uma das suas classes-base, deve-se utilizar o operador As.

Os operadores Is e As, bem como a conversão explícita entre classes, serão apresentados nos próximos tópicos.

O Operador Is

Existem situações em que, durante a execução de um programa, você precisa verificar se um objeto é uma instância de uma determinada classe, ou de uma das classes dela derivadas por herança.

Nesse caso, você pode utilizar o operador Is, usando a seguinte sintaxe:

```
<nome_objeto> Is <nome_classe>

Considere, a título de exemplificação, o seguinte trecho de código:

if (nome_objeto Is nome_classe)
     then
     // Código executado se a expressão é verdadeira
     else
     // Código executado se a expressão é falsa
```

Nesse caso, se o objeto cujo nome é "nome_objeto" for uma instância da classe "nome_classe" ou de uma das classes dela derivadas por herança, a expressão será verdadeira, e retornará o valor true.

É evidente, portanto, que se "nome_classe" for igual a TObject, essa expressão retornará sempre o valor true, independente do objeto analisado.

O Operador As

O operador As permite que um objeto de uma determinada classe seja tratado como uma instância de uma das suas classes-base.

A utilização do operador As segue esta sintaxe:

```
<nome_objeto> As <nome_classe>
```

Em geral, o operador As é utilizado para acessar um método ou propriedade da classe-base.

Nesse caso, deve-se colocar a expressão anterior entre parênteses, como indicado a seguir:

```
(nome_objeto As nome_classe).nome_método
(nome_objeto As nome_classe).nome_propriedade
```

Considere, por exemplo, que você queira limpar o texto exibido em todos os controles do tipo TEdit e TDBEdit existentes em um formulário, quando o usuário selecionar um botão de comando. Essa situação é comum, por exemplo, em formulários de cadastro de clientes, de produtos, fornecedores ou qualquer outra entidade.

Para obter esse resultado, basta definir da seguinte maneira o procedimento associado ao evento OnClick desse botão de comando:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   i:integer;
```

begin

```
  for i:= 0 to Componentcount-1 do
    if Components[i] is TCustomEdit then (Components[i] as TCustomEdit).Text := '';
end;
```

Nesse código, executa-se um "looping" pelos componentes inseridos no formulário (definidos pela sua propriedade Components). Para cada um dos componentes, verifica-se se o mesmo é um objeto de uma classe derivada da classe TCustomEdit, usando-se para isso o operador Is. Caso o componente seja um objeto de uma classe derivada da classe TCustomEdit, utiliza-se o operador As para efetuar uma conversão de tipo e atribuir uma string nula à sua propriedade Text.

Conversão Explícita Entre Tipos

Como alternativa ao operador As pode-se empregar a conversão explícita entre tipos, através da qual um objeto é tratado como uma instância de uma das suas classes-base.

A conversão explícita de tipos segue a seguinte sintaxe:

```
(Classe-Base)nome_objeto
```

Em geral, a conversão explícita entre tipos é utilizada para acessar um método ou propriedade da classe-base.

Nesse caso, deve-se colocar a expressão anterior entre parênteses, como indicado a seguir:

```
((Classe-Base)nome_objeto).nome_método
((Classe-Base)nome_objeto).nome_propriedade
```

O código apresentado no tópico anterior poderia, portanto, ser reescrito da seguinte maneira:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   i:integer;
begin
  for i:= 0 to Componentcount-1 do
     if Components[i] is TCustomEdit then (TCustomEdit(Components[i])).Text := '';
end;
```

O Identificador Self

O identificador Self pode ser utilizado para referenciar a instância corrente de uma classe em um dos seus métodos (é semelhante ao identificador this do C++).

Por incrível que pareça, isso é praticamente tudo o que você precisa saber sobre esse identificador. Diversos exemplos de utilização desse identificador serão apresentados ao longo do livro, o que tornará ainda mais fácil a compreensão do seu significado.

KNOW-HOW em: Tratamento de Exceções

PRÉ-REQUISITOS

- Conhecimento da sintaxe básica da linguagem Pascal e dos fundamentos da programação orientada a objetos em Delphi 7.

METODOLOGIA

- Apresentação do conceito de Exceção e da classe Exception.

Técnica

- Discussão das propriedades e métodos da classe Exception e das principais classes dela derivadas por herança.
- Implementação de novas exceções.

O Conceito de Exceções

Exceções são erros que ocorrem durante a execução do seu programa, mas que podem ser tratadas em um bloco de código do tipo try…except…end. Dessa maneira, seu código pode "tratar" os erros ocorridos durante a execução do programa, evitando que a execução da aplicação seja encerrada de forma brusca.

Para proteger um determinado trecho de código implementado na sua aplicação, deve-se utilizar um bloco try…except…end, que apresenta a seguinte sintaxe:

```
try
// Bloco de código a ser executado
except
// Bloco de código a ser executado caso haja algum erro na execução do bloco de
// código anterior
end;
```

Considere, por exemplo, um formulário no qual existam duas caixas de texto, denominadas EditNum1 e EditNum2, nas quais devem ser digitados dois números reais.

Além disso, considere que esse formulário possui um botão de comando no qual foi digitado o seguinte código no procedimento associado ao seu evento OnClick:

```
ShowMessage(FloatToStr(StrToFloat(EditNum1.Text)/StrToFloat(EditNum2.Text)));
```

Esse código exibe em uma mensagem o resultado da divisão dos valores digitados nas duas caixas de texto.

Caso você execute essa aplicação, digite o valor 4 na primeira caixa de texto e 0 na segunda; ao selecionar o botão de comando será gerada uma mensagem de erro com o texto:

```
Floating point division by zero.
```

Experimente, no entanto, redefinir o trecho de código do procedimento associado ao evento OnClick do botão de comando da seguinte maneira:

```
try
   ShowMessage(FloatToStr(StrToFloat(EditNum1.Text)/StrToFloat(EditNum2.Text)));
except
   ShowMessage('Você digitou valores incorretos');
   EditNum1.Text := '';
   EditNum2.Text := '';
end;
```

Nesse caso, se ocorrer algum problema durante a execução do programa, seja na conversão das strings em valores numéricos ou na divisão dos números, uma mensagem mais amigável será exibida. Além disso, o conteúdo das caixas de texto será apagado.

Você pode, no entanto, definir um tratamento para cada tipo de erro. Para isso, basta redefinir da seguinte maneira o código do procedimento associado ao evento OnClick do botão de comando:

```
try
     ShowMessage(FloatToStr(StrToFloat(EditNum1.Text)/StrToFloat(EditNum2.Text)));
except
     On EZeroDivide do
     begin
            ShowMessage('Você digitou valores incorretos');
            EditNum1.Text := '';
            EditNum2.Text := '';
     end;
     On EconvertError do
     begin
            ShowMessage('Erro de Conversão');
            EditNum1.Text := '';
```

```
            EditNum2.Text := '';
      end;
end;
```

Nesse caso, verifica-se o tipo de erro usando-se a cláusula:

```
on <tipo de exceção> do
```

Com essa cláusula, você pode definir uma resposta para cada um dos diversos tipos de erros que possam ser gerados durante a execução do seu aplicativo.

Referência: a Classe Exception

A classe Exception, codificada na unit SysUtils e reproduzida a seguir, é derivada por herança da classe TObject e é a classe-base para todas as classes que manipulam exceções no Delphi 7.

Repare que essa classe não segue a convenção padrão do Delphi 7, pela qual o nome de uma classe deve começar com um "T".

Definição da Classe Exception

Reproduzimos a seguir o trecho de código responsável pela definição da classe Exception.

```
Exception = class(TObject)
  private
    FMessage: string;
    FHelpContext: Integer;
  public
    constructor Create(const Msg: string);
    constructor CreateFmt(const Msg: string; const Args: array of const);
    constructor CreateRes(Ident: Integer; Dummy: Extended = 0);
    constructor CreateResFmt(Ident: Integer; const Args: array of const);
    constructor CreateHelp(const Msg: string; AHelpContext: Integer);
    constructor CreateFmtHelp(const Msg: string; const Args: array of const;
      AHelpContext: Integer);
    constructor CreateResHelp(Ident: Integer; AHelpContext: Integer);
    constructor CreateResFmtHelp(Ident: Integer; const Args: array of const;
      AHelpContext: Integer);
    property HelpContext: Integer read FHelpContext write FHelpContext;
    property Message: string read FMessage write FMessage;
  end;
```

Campos Internos da Classe Exception

Conforme pode ser verificado na sua definição, a classe Exception possui os seguintes campos internos, cuja visibilidade é definida como private:

```
FMessage: string;
FHelpContext: Integer;
```

FMessage

Esse campo, do tipo string, define o texto a ser exibido na caixa de diálogo que é mostrada quando ocorre um erro definido por essa exceção.

FHelpContext

Esse campo, do tipo inteiro, armazena o valor que identifica o tópico do arquivo de help on-line associado a essa classe.

Métodos Públicos da Classe Exception

Create

Declaração

```
constructor Create (const Msg: string);
```

Esse é um dos métodos construtores da classe (responsável pela alocação da memória necessária aos objetos da classe), e recebe como parâmetro uma string que será mostrada na caixa de diálogo exibida quando a exceção ocorrer durante a execução do programa.

Como já foi descrito anteriormente, uma classe pode definir mais de um método construtor, o que ocorre justamente na definição da classe Exception.

Como pode ser verificado na codificação do método (reproduzida a seguir), o valor passado como parâmetro no construtor da classe é armazenado no campo interno FMessage.

```
constructor Exception.Create(const Msg: string);
begin
  FMessage := Msg;
end;
```

CreateFmt

Declaração

```
constructor CreateFmt (const Msg: string; const Args: array of const)
```

Este é outro método construtor da classe (responsável pela alocação da memória necessária aos objetos da classe), e recebe como parâmetros uma string que será mostrada na caixa de diálogo exibida quando a exceção ocorrer durante a execução do programa, e um array contendo códigos de formatação para essa string.

Como pode ser verificado na codificação do método (reproduzida a seguir), os valores passados como parâmetros nesse construtor da classe são passados como parâmetros em uma chamada da função Format, e a string resultante da execução dessa função é armazenada no campo interno FMessage.

```
constructor Exception.CreateFmt(const Msg: string;
  const Args: array of const);
begin
  FMessage := Format(Msg, Args);
end;
```

Os códigos de formatação são apresentados no item “Format Strings”do Help On-line do Delphi 7.

CreateFmtHelp

Declaração

```
constructor CreateFmtHelp (const Msg: string; const Args: array of const; AHelpContext:
Integer);
```

Este é outro método construtor da classe (responsável pela alocação da memória necessária aos objetos da classe), e recebe como parâmetros uma string que será mostrada na caixa de diálogo exibida quando a exceção ocorrer durante a execução do programa, um array contendo códigos de formatação para essa string e um inteiro identificando o tópico do arquivo de help on-line a ser associado à classe.

Como pode ser verificado na codificação do método (reproduzida a seguir), os dois primeiros valores passados como parâmetros nesse construtor da classe são passados como parâmetros em uma chamada da função Format, e a string resultante da execução dessa função é armazenada no campo interno FMessage. O terceiro parâmetro, por sua vez, é armazenado no campo interno FhelpContext.

```
constructor Exception.CreateFmtHelp(const Msg: string; const Args: array of const;
  AHelpContext: Integer);
begin
  FMessage := Format(Msg, Args);
  FHelpContext := AHelpContext;
end;
```

CreateHelp

Declaração

```
constructor CreateHelp (const Msg: string; AHelpContext: Integer);
```

Este é outro método construtor da classe (responsável pela alocação da memória necessária aos objetos da classe), e recebe como parâmetros uma string que será mostrada na caixa de diálogo exibida quando a exceção ocorrer durante a execução do programa e um inteiro identificando o tópico do arquivo de help on-line a ser associado à classe.

A diferença entre esse construtor e o descrito no tópico anterior está no fato de que a string não é formatada, não havendo portanto uma chamada à função Format.

A codificação deste método é reproduzida a seguir.

```
constructor Exception.CreateHelp(const Msg: string; AHelpContext: Integer);
begin
  FMessage := Msg;
  FHelpContext := AHelpContext;
end;
```

CreateRes

Declaração

```
constructor CreateRes (Ident: Integer; Dummy: Extended = 0);
```

Este é outro método construtor da classe (responsável pela alocação da memória necessária aos objetos da classe), e recebe como parâmetro um inteiro que identifica uma string armazenada no arquivo de recursos do seu aplicativo.

Nesse caso, a string a ser armazenada no campo interno FMessage é obtida executando-se a função LoadStr, que retorna a string armazenada no arquivo de recursos identificada pelo valor passado como parâmetro.

A codificação deste método é reproduzida a seguir.

```
constructor Exception.CreateRes(Ident: Integer);
begin
  FMessage := LoadStr(Ident);
end;
```

CreateResFmt

Declaração

```
constructor CreateResFmt(Ident: Integer; const Args: array of const);
```

Esse é outro método construtor da classe (responsável pela alocação da memória necessária aos objetos da classe), e recebe como parâmetros um inteiro que identifica uma string armazenada no arquivo de recursos do seu aplicativo e um array contendo códigos de formatação para essa string.

Nesse caso, a string a ser armazenada no campo interno FMessage é obtida executando-se a função Format, passando como parâmetro a string resultante de uma chamada à função LoadStr (que retorna a string armazenada no arquivo de recursos identificada pelo valor passado como parâmetro).

A codificação deste método é reproduzida a seguir.

```
constructor Exception.CreateResFmt(Ident: Integer;
  const Args: array of const);
begin
  FMessage := Format(LoadStr(Ident), Args);
end;
```

CreateResFmtHelp

Declaração

```
constructor CreateResFmtHelp(Ident: Integer; const Args: array of const
; AHelpContext: Integer);
```

Este é outro método construtor da classe (responsável pela alocação da memória necessária aos objetos da classe), e recebe como parâmetros um inteiro que identifica uma string armazenada no arquivo de recursos do seu aplicativo, um array contendo códigos de formatação para essa string e um valor inteiro que identifica o código do arquivo de help associado a essa classe.

A diferença entre esse método e o descrito no tópico anterior está na inclusão do terceiro parâmetro, que identifica o código do arquivo de help associado à classe, e cujo valor será armazenado no campo interno FhelpContext.

A codificação deste método é reproduzida a seguir.

```
constructor Exception.CreateResFmtHelp(Ident: Integer;
  const Args: array of const;
  AHelpContext: Integer);
begin
```

```
    FMessage := Format(LoadStr(Ident), Args);
    FHelpContext := AHelpContext;
  end;
```

Propriedades da Classe Exception

HelpContext

A propriedade HelpContext é definida pela seguinte linha de código:

```
property HelpContext: Integer read FHelpContext write FHelpContext;
```

Conforme pode ser verificado, o valor atribuído à propriedade HelpContext é diretamente armazenado no campo FHelpContext. Da mesma forma, ao se tentar obter o valor dessa propriedade, esse também é obtido diretamente do campo FHelpContext.

Repare que, nesse caso, não se definiu nenhum método de leitura ou de escrita, e conseqüentemente a propriedade acessa diretamente o valor armazenado em um campo interno da classe.

Message

A propriedade Message é definida pela seguinte linha de código.

```
property Message: string read FMessage write FMessage;
```

Conforme pode ser verificado, o valor atribuído à propriedade Message é diretamente armazenado no campo FMessage. Da mesma forma, ao se tentar obter o valor dessa propriedade, esse também é obtido diretamente do campo FMessage.

Repare que, nesse caso, também não se definiu nenhum método de leitura ou de escrita, e conseqüentemente a propriedade acessa diretamente o valor armazenado em um campo interno da classe.

Classes Derivadas por Herança da Classe Exception

A classe Exception é a classe-base de todas as classes criadas para manipular exceções.

O Delphi 7 define várias classes destinadas a manipulação de exceções (derivadas de Exception), como pode ser verificado no código-fonte da unit SysUtils.

Muitas dessas classes, no entanto, não implementam nenhum campo adicional ou sobrecarregam qualquer dos métodos definidos na classe-base.

Nesses casos, essas classes servem apenas como um "alias" ou apelido para a classe Exception, a serem utilizadas em determinadas situações.

Nó próximo tópico será apresentada a sintaxe necessária à geração de uma exceção.

O Mecanismo Utilizado na Geração de Exceções

Para gerar uma exceção deve-se usar o comando raise, seguido pela execução do método construtor da classe correspondente à exceção a ser gerada.

Considere, por exemplo, a definição da seguinte exceção, denominada ERendimento Error:

```
ERendimento Error = Class(Exception);
```

Essa classe pode ser usada para exibir uma mensagem de erro para o usuário da classe, sempre que se tentar atribuir um valor incorreto para o campo.

Para isso, podemos redefinir a classe TContribuinte (já definida anteriormente) da maneira mostrada a seguir.

```
TContribuinte = Class
   private
      // Campos e métodos privados devem ser definidos aqui.
      FRendimento : double;
      Fimposto : double;
   public
      procedure SetRendimento (valor : double);
      function calcula_imposto:double;
      property rendimento : double read FRendimento write SetRendimento;
      property imposto : double read calcula_imposto;
   protected
      // Campos e métodos protegidos devem ser definidos aqui.
end;
.....

function TContribuinte.calcula_imposto:double;
begin
     if (FRendimento < 900.0) then Fimposto := 0.0;
     if (FRendimento > 1800) then Fimposto := 0.25*rendimento -315.0;
     if (FRendimento >= 900)and(FRendimento <= 1800) then Fimposto := 0.15*rendimento -135.0;
     result := Fimposto;
end;

procedure TContribuinte.SetRendimento (Valor : double);
begin
if valor < 0
     then
     raise ERendimentoError.Create('Rendimento Não pode ser Negativo')
     else
            FRendimento := valor;
end;
```

Dessa maneira, se você tentar atribuir um valor negativo à propriedade rendimento de uma instância da classe TContribuinte, será exibida uma caixa de diálogo com a mensagem 'Rendimento Não pode ser Negativo' (ainda bem!).

As exceções permitem, portanto, que se faça um tratamento diferenciado em função do tipo de erro que provoca uma interrupção na execução normal de um programa.

Você pode tratar o erro descrito anteriormente em um bloco try..except semelhante ao mostrado no seguinte trecho de código, onde valor é uma variável do tipo double:

```
try
          contribuinte1.rendimento := valor;
     except
          on ERendimento Error do
            ShowMessage('Rendimento Não pode ser Negativo');
     end;
```

KNOW-HOW em: Manipulação de Listas de Objetos

PRÉ-REQUISITOS

- Conhecimento da sintaxe básica da linguagem Pascal e dos fundamentos da programação orientada a objetos em Delphi 7.

METODOLOGIA

- Apresentação da classe TList, que permite a manipulação de listas de objetos.

TÉCNICA

- Implementação de classes derivadas da classe TList, de forma a ilustrar o mecanismo de manipulação de listas de objetos.

Apresentação do Problema

Considere a seguinte situação: sua aplicação define uma determinada classe, e precisa manipular uma lista de objetos dessa classe.

Vamos considerar novamente a classe TPessoa_Fisica, derivada da classe TContribuinte, à qual será acrescida o campo nome, do tipo string, como a seguir:

```
TPessoa_Fisica = Class(TContribuinte)
     nome: string;
end;
```

Caso a sua aplicação necessite manipular uma lista de contribuintes do tipo pessoa física, a princípio seria necessário criar uma nova classe com essa finalidade.

A fim de facilitar o trabalho do desenvolvedor, o Delphi 7 inclui entre as diversas classes disponíveis a classe TList, que declara a maioria dos métodos necessários à manipulação de listas de objetos.

É importante, no entanto, ressaltar que uma instância de uma classe armazena, na realidade, um endereço para um objeto da classe, e não o objeto propriamente dito – embora o Delphi 7 torne o acesso ao objeto da classe tão transparente que muitos consideram que o objeto esteja sendo acessado diretamente. Quando manipulamos um objeto (ou instância) de uma classe estamos na realidade acessando o endereço ocupado pelo objeto – a questão é que o Delphi 7 trata de desreferenciar o ponteiro.

Já que o ambiente de desenvolvimento tratou de simplificar essa tarefa, não vamos nos deter em explicar a aritmética de ponteiros definida na linguagem Object Pascal – seria complicar aquilo que já foi simplificado – na contramão da produtividade desejada por aqueles que utilizam o Delphi 7 como ferramenta de desenvolvimento.

É importante, no entanto, que esse conceito fique bem claro pelo leitor: embora tudo se passe como se estivéssemos acessando diretamente uma instância da classe, o nome do objeto armazena na realidade o endereço do objeto. Conforme será visto a seguir, a classe TList manipula endereços de objetos, e não armazena os objetos propriamente ditos.

Podemos portanto pensar em criar a classe TListaContribuinte a partir da classe TList, aproveitando as características já disponíveis para essa classe.

Apresentamos a seguir a descrição das principais propriedades, métodos e eventos da classe TList.

Referência: a Classe TList

A classe TList, codificada na unit Classes e reproduzida a seguir, é derivada por herança da classe TObject e é a classe-base para todas as classes que manipulam listas de objetos no Delphi 7.

Definição da Classe TList

Reproduzimos a seguir o trecho de código responsável pela definição da classe TList.

```
TList = class(TObject)
  private
    FList: PPointerList;
    FCount: Integer;
    FCapacity: Integer;
  protected
    function Get(Index: Integer): Pointer;
    procedure Grow; virtual;
    procedure Put(Index: Integer; Item: Pointer);
    procedure SetCapacity(NewCapacity: Integer);
    procedure SetCount(NewCount: Integer);
  public
    destructor Destroy; override;
    function Add(Item: Pointer): Integer;
    procedure Clear; dynamic;
    procedure Delete(Index: Integer);
    class procedure Error(const Msg: string; Data: Integer); virtual;
    procedure Exchange(Index1, Index2: Integer);
    function Expand: TList;
    function First: Pointer;
    function IndexOf(Item: Pointer): Integer;
    procedure Insert(Index: Integer; Item: Pointer);
    function Last: Pointer;
    procedure Move(CurIndex, NewIndex: Integer);
    function Remove(Item: Pointer): Integer;
    procedure Pack;
    procedure Sort(Compare: TListSortCompare);
    property Capacity: Integer read FCapacity write SetCapacity;
    property Count: Integer read FCount write SetCount;
    property Items[Index: Integer]: Pointer read Get write Put; default;
    property List: PPointerList read FList;
  end;
```

Propriedades da Classe TList

Apresentamos a seguir a descrição das propriedades definidas para a classe TList.

Capacity

Essa propriedade é declarada como uma variável inteira, e retorna o número de objetos que podem ser referenciados na lista. Essa propriedade retorna o número de objetos que a lista é capaz de referenciar, e não o número de objetos que realmente estão sendo endereçados pela lista, o que é definido pela sua propriedade Count (a ser vista a seguir).

Inicialmente (quando a lista é criada) o seu valor é igual a 0, mas é alterado durante a manipulação da lista.

Count

Essa propriedade é declarada como uma variável inteira, e retorna o número de objetos referenciados na lista e não o número de objetos que podem vir a ser endereçados pela mesma, o que é definido pela sua propriedade Capacity, descrita anteriormente.

O valor da propriedade Count pode ser inferior ao definido pela propriedade Capacity, pois a lista pode, eventualmente, endereçar ponteiros nulos (que armazenam o valor nil). Para remover essas referências a endereços nos quais não está armazenado nenhum objeto, deve-se utilizar o método Pack da classe, a ser descrito posteriormente.

Items

Essa propriedade armazena, na forma de um array, as referências a todos os objetos endereçados pela lista.

Esse array inicia com o índice 0 e, conseqüentemente, o endereço do primeiro objeto referenciado pela lista estará armazenado em Items[0].

List

Essa propriedade armazena, na forma de um array, os endereços de todos os objetos da lista (as variáveis do tipo Pointer que armazenam os endereços dos objetos referenciados pela lista).

Principais Métodos da Classe TList

Add

Declaração

```
function Add(Item: Pointer): Integer;
```

Esse método adiciona uma referência a um objeto no final da lista, e retorna um valor inteiro que retorna o índice dessa referência no array (esse índice começa em zero).

Clear

Declaração

```
procedure Clear; dynamic;
```

Esse método remove todas as referências a objetos da lista, e atribui o valor 0 às propriedades Count e Capacity.

Delete

Declaração

```
procedure Delete(Index: Integer);
```

Esse método remove da lista o endereço do objeto cujo índice é passado como parâmetro. Conseqüentemente, o índice das referências a objetos cujo índice atual é superior ao da referência removida da lista tem o seu valor reduzido em uma unidade.

Destroy

Declaração

```
destructor Destroy; override;
```

Este é o método destrutor da classe, responsável por remover todas as referências a objetos da lista. Aconselha-se utilizar o método Free (herdado da classe TObject) em vez do método Destroy, pois, conforme já descrito anteriormente, o método Free verifica se o valor do endereço é igual a nil antes de chamar o método destrutor da classe.

É importante destacar que esse método não destrói os objetos propriamente ditos, mas sim as suas referências na lista. Para destruir cada objeto, deve-se chamar seu próprio método construtor.

Error

Declaração

```
class procedure Error(const Msg: string; Data: Integer); virtual;
```

Este método deve ser usado para gerar uma exceção caso ocorra algum erro durante a manipulação dos objetos da lista, e recebe como parâmetros uma string e um valor inteiro que pode ser usado para indicar a causa do erro.

Exchange

Declaração

```
procedure Exchange(Index1, Index2: Integer);
```

Esse método troca a posição na lista das referências a dois objetos cujos índices são passados como parâmetros.

Expand

Declaração

```
function Expand: TList;
```

Esse método expande a capacidade da lista.

Se Count = Capacity, o valor da propriedade Capacity é incrementado de acordo com as seguintes regras:

- Se o valor da propriedade Capacity é maior do que 8, seu valor é incrementado em 16 unidades.
- Se o valor da propriedade Capacity é maior do que 4 e menor ou igual a 8, seu valor é incrementado em 8 unidades.
- Se o valor da propriedade Capacity é menor do que 4, seu valor é incrementado em 4 unidades.

First

Declaração

```
function First: Pointer;
```

Esse método retorna em um ponteiro o endereço do primeiro objeto referenciado pela lista.

IndexOf

Declaração

```
function IndexOf(Item: Pointer): Integer;
```

Esse método retorna o índice da referência a objeto cujo endereço é passado como parâmetro.

Insert

Declaração

```
procedure Insert(Index: Integer; Item: Pointer);
```

Esse método insere, na posição da lista definida pelo índice cujo valor é passado como primeiro parâmetro, o endereço do objeto a ser referenciado, que é passado como segundo parâmetro na forma de um ponteiro.

Conseqüentemente, o índice dos objetos cujo índice atual é igual ou superior ao do objeto inserido na lista tem o seu valor incrementado em uma unidade.

Last

Declaração

```
function Last: Pointer;
```

Esse método retorna em um ponteiro o endereço do último objeto referenciado pela lista.

Move

Declaração

```
procedure Move(CurIndex, NewIndex: Integer);
```

Esse método move a referência ao objeto definida pelo índice passado como primeiro parâmetro para a posição indicada pelo índice passado como segundo parâmetro.

Pack

Declaração

```
procedure Pack;
```

Esse método remove todos os itens que não referenciam qualquer objeto, isto é, que armazenam o valor nil, para o início da lista e reduz o valor da propriedade Count para o número de objetos realmente referenciados pela lista.

Esse método, no entanto, não libera a memória usada para armazenar os ponteiros cujo valor é nil (o que pode ser feito atribuindo-se à propriedade Capacity o valor armazenado na propriedade Count).

Remove

Declaração

```
function Remove(Item: Pointer): Integer;
```

Esse método remove da lista a referência ao objeto cujo endereço é passado como parâmetro na forma de um ponteiro, retornando o índice do objeto cuja referência na lista foi removida. Conseqüentemente, o índice das referências a objetos cujo índice atual é superior ao do objeto removido da lista tem o seu valor reduzido em uma unidade.

Sort

Declaração

```
procedure Sort(Compare: TListSortCompare);
```

Esse método ordena os itens da lista usando o algoritmo QuickSort, usando como valor para definir essa ordenação o retornado pela função passada como parâmetro.

Essa função deve receber como parâmetros dois ponteiros que referenciam itens da lista e retorna:

- Um valor negativo (< 0), se o objeto referenciado por item1 é, segundo o critério de ordenação definido no corpo da função, menor do que o objeto referenciado por item2.
- Um valor nulo (= 0), se o objeto referenciado por item1 é, segundo o critério de ordenação definido no corpo da função, igual ao objeto referenciado por item2.
- Um valor positivo (> 0), se o objeto referenciado por item1 é, segundo o critério de ordenação definido no corpo da função, maior do que o objeto referenciado por item2.

O cabeçalho da função usada na ordenação dos itens da lista é definido pelo tipo TListSortCompare, indicado a seguir.

```
type TListSortCompare = function (Item1, Item2: Pointer): Integer;
```

Exemplo de Utilização

Nesse tópico apresentaremos um exemplo de utilização de uma classe derivada da classe TList.

Definição da Interface

Inicialmente deverá ser definida uma nova aplicação, composta por um formulário no qual serão inseridos vários componentes, com as propriedades descritas a seguir:

- Formulário:

  Name: FormListaObjetos
  Caption: Formulário Para a Manipulação de Listas de Objetos
  Position: poScreenCenter

- Botão de Figura:

  Name: BotaoFechar
  Kind: bkClose
  Caption: &Fechar

- GroupBoxes:

  Name: GroupBoxDados
  Caption: Dados do Contribuinte

  Name: GroupBoxOperacoes
  Caption: Operações Sobre a Lista

  Name: GroupBoxPesquisa
  Caption: Digite o Nome do Contribuinte a Pesquisar

Objetos a serem inseridos dentro do GroupBox GroupBoxDados:

- Labels:

  Name: LabelNome
  Caption: Nome:

  Name: LabelRendimento
  Caption: Rendimento:

  Name: LabelImposto
  Caption: Imposto:

  Name: LabelValorImposto
  Caption: 0.0

- Caixas de Texto:

  Name: EditNome
  Text:
  Name: EditRendimento
  Text:

Objetos a serem inseridos dentro do GroupBox GroupBoxOperacoes:

- Botões de Comando:

  Name: BotaoAdiciona
  Caption: Adiciona

Name: BotaoInsere
Caption: Insere

Name: BotaoRemove
Enabled: False
Caption: Remove

Name: BotaoOrdena
Caption: Ordena

Name: BotaoPrimeiro
Caption: Primeiro

Name: BotaoAnterior
Caption: Anterior

Name: BotaoProximo
Caption: Próximo

Name: BotaoUltimo
Caption: Último

Objetos a serem inseridos dentro do GroupBox GroupBoxPesquisa:

- Caixas de Texto:

  Name: EditPesquisa
  Text:

Reposicione e redimensione estes componentes para que o formulário fique com o aspecto mostrado na figura a seguir.

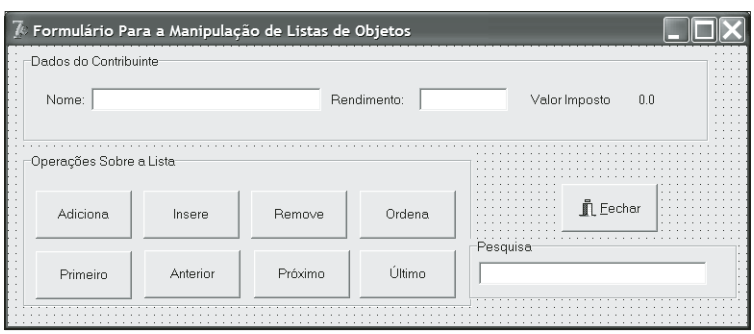


Figura 19.5: Aspecto do formulário usado no exemplo de manipulação de listas.

Codificação do Exemplo

Será criada uma classe chamada TListaContribuinte, destinada a manipular uma lista de objetos da classe TPessoa_Fisica, na qual serão incluídos métodos não existentes em sua classe ancestral (TList).

A definição dessa classe deve ser feita na seção type da unit associada a esse formulário, mediante a inclusão das seguintes linhas de código:

```
TContribuinte = Class
   private
      // Campos e métodos privados devem ser definidos aqui.
      FRendimento : double;
      Fimposto : double;
   public
      procedure SetRendimento (valor : double);
      function calcula_imposto:double;
      property rendimento : double read FRendimento write SetRendimento;
      property imposto : double read calcula_imposto;
   protected
      // Campos e métodos protegidos devem ser definidos aqui.
   end;

   ERendimento = Class(Exception);

   TPessoa_Fisica = class(TContribuinte)
     nome: string;
  end;

TListaContribuinte = class(TList)
    procedure PesquisaNome(Nome:String);
    destructor Destroy;override;
  end;
```

Repare que incluímos ainda as definições das classes TContribuinte (ancestral da classe TPessoa_Fisica) e ERendimentoError (exceção manipulada pela classe TContribuinte).

Repare ainda que, nesse caso, optamos por manter o campo rendimento definido na classe TContribuinte, de maneira a aproveitar o código já escrito em tópicos anteriores.

O único campo acrescentado na definição da classe TPessoa_Fisica foi o campo nome, do tipo string.

A classe TListaContribuinte, derivada por herança da classe TList, acrescenta na sua definição os métodos PesquisaNome e Destroy (que sobrecarrega o método de mesmo nome definido na classe-base).

Para criar o corpo principal de um método da classe, basta selecionar simultaneamente as teclas Ctrl, Shift e C após digitar as declarações dos métodos na definição da classe. O ambiente de desenvolvimento vai gerar automaticamente o corpo principal dos métodos na seção Implementation da unit.

Declare as seguintes variáveis na seção var da unit:

```
ListaContribuinte : TListaContribuinte;
indiceatual : integer;
```

O objeto ListaContribuinte, da classe TListaContribuinte, será usado para representar a lista que manipulará os objetos.

A variável indiceatual, do tipo inteiro, será usada para armazenar o índice do objeto cujos valores são exibidos no momento.

Para alocar a memória necessária ao objeto ListaContribuinte, deve-se fazer uma chamada ao seu método construtor no procedimento associado ao evento OnCreate do formulário, que deve ser codificado da seguinte maneira:

```
procedure TFormListaObjetos.FormCreate(Sender: TObject);
begin
     ListaContribuinte := TListaContribuinte.Create;
end;
```

Para liberar a memória alocada ao término da execução da aplicação, basta definir da seguinte maneira o procedimento associado ao evento OnDestroy do formulário:

```
procedure TFormListaObjetos.FormDestroy(Sender: TObject);
begin
     ListaContribuinte.Free;
end;
```

Para permitir ao usuário adicionar um elemento à lista, deve-se definir da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoAdicionar:

```
procedure TFormListaObjetos.BotaoAdicionaClick(Sender: TObject);
var
   Contribuinte: TPessoa_Fisica;
begin
   Contribuinte := TPessoa_Fisica.Create;
   Contribuinte.Nome := EditNome.Text;
   Contribuinte.rendimento := StrToFloat(EditRendimento.Text);
   ListaContribuinte.Add(Contribuinte);
   indiceatual := ListaContribuinte.Count-1;
LabelValorImposto.Caption:=FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
BotaoRemove.Enabled := True;
end;
```

Repare que esse procedimento declara uma instância local da classe TPessoa_Fisica, e que a chamada ao seu método Create retorna uma nova instância da classe, o que é feito mediante a inclusão da seguinte linha de código:

```
Contribuinte := TPessoa_Fisica.Create;
```

Em seguida, atribuem-se valores aos campos Nome e rendimento da classe, o que é feito mediante a inclusão das seguintes linhas de código:

```
Contribuinte.Nome := EditNome.Text;
Contribuinte.rendimento := StrToFloat(EditRendimento.Text);
```

A adição do objeto à lista é feita mediante uma chamada ao método Add da classe TListaContribuinte, mediante a inclusão da seguinte linha de código:

```
ListaContribuinte.Add(Contribuinte);
```

Por fim, o valor do índice atual e o texto exibido no label LabelValorImposto são atualizados, o que é feito mediante a inclusão das seguintes linhas de código:

```
indiceatual := ListaContribuinte.Count-1;
LabelValorImposto.Caption:=FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
```

Repare a conversão explícita de tipos empregada nesse código. Isso se deve ao fato de que a propriedade Items da classe TList (e conseqüentemente da classe TListaContribuinte) nos fornece os endereços dos objetos manipulados pela lista e, como sabemos que os objetos são da classe TPessoa_Fisica, podemos realizar essa conversão com segurança.

Para permitir que o usuário possa remover posteriormente os objetos adicionados à lista, deve-se habilitar o componente BotaoRemove, o que é feito mediante a inclusão da seguinte linha de código:

```
BotaoRemove.Enabled := True;
```

Para permitir ao usuário inserir um elemento na lista, deve-se definir da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoInsere:

```
procedure TFormListaObjetos.BotaoInsereClick(Sender: TObject);
var
   Contribuinte: TPessoa_Fisica;
begin
   Contribuinte := TPessoa_Fisica.Create;
   Contribuinte.Nome := EditNome.Text;
   Contribuinte.rendimento := StrToFloat(EditRendimento.Text);
   ListaContribuinte.Insert(indiceatual, Contribuinte);
  BotaoRemove.Enabled := True;
end;
```

Esse código é muito semelhante ao definido para o procedimento associado ao evento OnClick do componente BotaoAdiciona, exceto que se utiliza nesse caso o método Insert em vez do método Add da classe TListaContribuinte.

Para permitir ao usuário remover um elemento na lista, deve-se definir da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoRemove:

```
procedure TFormListaObjetos.BotaoRemoveClick(Sender: TObject);
begin
     ListaContribuinte.Delete(indiceatual);
     if indiceatual > ListaContribuinte.Count-1 then indiceatual :=
     ListaContribuinte.Count-1;
     if ListaContribuinte.Count > 0 then
     begin
          EditNome.Text :=
          TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
EditRendimento.Text:=FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).
rendimento);
          LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
     end
     else
     begin
          EditNome.Clear;
          EditRendimento.Clear;
          LabelValorImposto.Caption := '';
       BotaoRemove.Enabled := False;
     end;
end;
```

É importante, inicialmente, verificar que esse procedimento só será executado se a propriedade Enabled do componente for igual a True.

Inicialmente remove-se o objeto atual da lista (e cujo índice é identificado pela variável indiceatual) mediante uma chamada ao método Delete da classe TListaContribuinte, o que é feito mediante a inclusão da seguinte linha de código:

```
ListaContribuinte.Delete(indiceatual);
```

Em seguida, verifica-se se o valor armazenado na variável índice atual está dentro dos limites possíveis para o índice do array, o que é feito mediante a inclusão da seguinte linha de código:

```
if indiceatual > ListaContribuinte.Count-1 then indiceatual := ListaContribuinte.Count-1;
```

Por fim, verifica-se se ainda existem elementos na lista (analisando-se o valor da propriedade Count do objeto ListaContribuinte), e atualizam-se corretamente os valores exibidos nas caixas de texto e no label LabelValorImposto, o que é feito mediante a inclusão do seguinte trecho de código:

```
if ListaContribuinte.Count > 0 then
     begin
          EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
EditRendimento.Text:=FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).
rendimento);
          LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
     end
     else
     begin
          EditNome.Clear;
          EditRendimento.Clear;
          LabelValorImposto.Caption := '';
BotaoRemove.Enabled := False;
     end;
```

Repare que, se não houver mais objetos na lista, o botão BotaoRemove é desabilitado, mediante a inclusão da seguinte linha de código:

```
Enabled := False;
```

Para permitir ao usuário exibir os dados armazenados no primeiro objeto da classe, deve-se definir da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoPrimeiro:

```
procedure TFormListaObjetos.BotaoPrimeiroClick(Sender: TObject);
begin
     ListaContribuinte.First;
     indiceatual := 0;
     EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
     EditRendimento.Text :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).rendimento);
     LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
end;
```

Inicialmente, faz-se uma chamada ao método First da classe TListaContribuinte, mediante a inclusão da seguinte linha de código:

```
ListaContribuinte.First;
```

'Em seguida atualizam-se o valor da variável indiceatual – que deve ser igual ao índice do primeiro elemento da lista – isto é, 0, o que é feito mediante a inclusão da seguinte linha de código:

```
indiceatual := 0;
```

Por fim, atualizam-se os valores exibidos nas caixas de texto e no label LabelValorImposto, o que é feito mediante a inclusão das seguintes linhas de código:

```
EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
EditRendimento.Text :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).rendimento);
LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
```

Os procedimentos associados ao evento OnClick dos componentes BotaoProximo, BotaoUltimo e BotaoAnterior têm codificação semelhante, sendo sua definição reproduzida a seguir.

```
procedure TFormListaObjetos.BotaoAnteriorClick(Sender: TObject);
begin
     if indiceatual > 0 then indiceatual := indiceatual-1;
     EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
     EditRendimento.Text := FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).
rendimento);
     LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
end;
procedure TFormListaObjetos.BotaoProximoClick(Sender: TObject);
begin
     if indiceatual < ListaContribuinte.Count-1 then indiceatual := indiceatual+1;
     EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
     EditRendimento.Text :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).rendimento);
     LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
end;

procedure TFormListaObjetos.BotaoUltimoClick(Sender: TObject);
begin
     ListaContribuinte.Last;
     indiceatual := ListaContribuinte.Count-1;
     EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
     EditRendimento.Text :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).rendimento);
     LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
end;
```

O único cuidado que se deve ter na codificação desses procedimentos se refere a garantir que os limites do índice do array sejam respeitados.

O método PesquisaNome da classe TListaContribuinte deve ser capaz de retornar o índice do objeto manipulado pela lista cujo valor armazenado no campo Nome seja igual ao da string passada como parâmetro.

Apresenta-se a seguir a codificação desse método:

```
procedure TListaContribuinte.PesquisaNome(Nome: String);
var
   i : integer;
```

```
begin
    First;
    indiceatual := 0;
    for i := 0 to Count-1 do
    begin
        if TPessoa_Fisica(Items[i]).Nome = Nome then
        begin
            indiceatual := i;
            Exit;
        end;
    end;
end;
```

Nesse procedimento, define-se uma variável local i a ser usada no loop de pesquisa.

Inicialmente define-se o objeto corrente como sendo o primeiro objeto manipulado pela lista, mediante uma chamada ao método First da classe TListaContribuinte, o que é feito mediante a inclusão da seguinte linha de código:

```
First;
```

Além disso, atribui-se o valor 0 à variável índice atual:

```
indiceatual := 0;
```

Por fim, efetua-se um "looping" pelos objetos referenciados pela lista, até que se encontre um objeto cujo valor armazenado no campo Nome é igual ao da string passada como parâmetro (atribuindo-se o valor atual do contador à variável indiceatual).

```
    for i := 0 to Count-1 do
    begin
        if TPessoa_Fisica(Items[i]).Nome = Nome then
        begin
            indiceatual := i;
            Exit;
        end;
    end;
```

Repare que nesse caso implementou-se uma pesquisa exata, e não uma pesquisa aproximada.

Para permitir ao usuário ordenar os objetos referenciados pela lista, deve-se definir da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoOrdena:

```
procedure TFormListaObjetos.BotaoOrdenaClick(Sender: TObject);
begin
    ListaContribuinte.Sort(Compara);
end;
```

O código desse procedimento consiste apenas em uma chamada ao método Sort da classe TListaContribuinte, o que é feito mediante a inclusão da seguinte linha de código:

```
ListaContribuinte.Sort(Compara);
```

Falta ainda definir a função a ser usada pelo algoritmo QuickSort. Nesse caso, definiu-se uma função denominada Compara, e implementada da seguinte maneira:

```
function Compara(indice1, indice2: Pointer): integer;
```

```
begin
result:=AnsiCompareText(TPessoa_Fisica(indice1).Nome,TPessoa_Fisica(indice2).Nome);
end;
```

Repare que essa função utiliza a função AnsiCompareText, definida na unit SysUtils, e que compara duas strings sem levar em consideração se a letra está em caixa alta ou caixa baixa.

Por fim, falta apresentar a definição do método destrutor da classe TListaContribuinte, cuja codificação é reproduzida a seguir:

```
destructor TListaContribuinte.Destroy;
var
   i : integer;
begin
     for i:= 0 to Count-1 do
     begin
         TPessoa_Fisica(Items[i]).Destroy;
     end;
end;
```

Repare que esse método é responsável por liberar a memória alocada para todos os objetos referenciados pela lista.

Para simplificar a digitação de novos valores pelo usuário, os procedimentos associados ao evento OnEnter das caixas de texto EditNome e EditRendimento devem ser implementados da seguinte maneira:

```
procedure TFormListaObjetos.EditNomeEnter(Sender: TObject);
begin
     EditNome.Clear;
     LabelValorImposto.Caption := '0.0';
end;
procedure TFormListaObjetos.EditRendimento Enter(Sender: TObject);
begin
     EditRendimento.Clear;
end;
```

Apresentamos a seguir o código completo da unit Ulista:

```
unit Ulista;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, StdCtrls;
type
  TFormListaObjetos = class(TForm)
    GroupBoxDados: TGroupBox;
    LabelNome: TLabel;
    EditNome: TEdit;
    LabelRendimento: TLabel;
    EditRendimento: TEdit;
    GroupBoxOperacoes: TGroupBox;
    BotaoAdiciona: TButton;
    BotaoInsere: TButton;
    BotaoRemove: TButton;
    BotaoOrdena: TButton;
    BotaoPrimeiro: TButton;
    BotaoAnterior: TButton;
    BotaoProximo: TButton;
    BotaoUltimo: TButton;
    BotaoFechar: TBitBtn;
```

```
    LabelImposto: TLabel;
    LabelValorImposto: TLabel;
    GroupBoxPesquisa: TGroupBox;
    EditPesquisa: TEdit;
    procedure FormCreate(Sender: TObject);
    procedure BotaoAdicionaClick(Sender: TObject);
    procedure BotaoInsereClick(Sender: TObject);
    procedure BotaoOrdenaClick(Sender: TObject);
    procedure BotaoPrimeiroClick(Sender: TObject);
    procedure BotaoAnteriorClick(Sender: TObject);
    procedure BotaoProximoClick(Sender: TObject);
    procedure BotaoUltimoClick(Sender: TObject);
    procedure EditPesquisaChange(Sender: TObject);
    procedure EditNomeEnter(Sender: TObject);
    procedure EditRendimento Enter(Sender: TObject);
    procedure BotaoRemoveClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TContribuinte = Class
   private
      // Campos e métodos privados devem ser definidos aqui.
      FRendimento : double;
      Fimposto : double;
   public
      procedure SetRendimento (valor : double);
      function calcula_imposto:double;
      property Rendimento : double read FRendimento write SetRendimento;
      property imposto : double read calcula_imposto;
   protected
      // Campos e métodos protegidos devem ser definidos aqui.
   end;

   ERendimento Error = Class(Exception);

   TPessoa_Fisica = class(TContribuinte)
     nome: string;
  end;

  TListaContribuinte = class(TList)
    procedure PesquisaNome(Nome:String);
    destructor Destroy;override;
  end;

  function Compara(indice1, indice2 : Pointer):integer;

var
  FormListaObjetos: TFormListaObjetos;
  ListaContribuinte : TListaContribuinte;
  indiceatual : integer;

implementation

{$R *.DFM}

{ TListaContribuinte }

function Compara(indice1, indice2: Pointer): integer;
begin
    result := AnsiCompareText(TPessoa_Fisica(indice1).Nome,TPessoa_Fisica(indice2).Nome);
```

```
end;

destructor TListaContribuinte.Destroy;
var
   i : integer;
begin
     for i:= 0 to Count-1 do
     begin
         TPessoa_Fisica(Items[i]).Destroy;
     end;
end;

procedure TListaContribuinte.PesquisaNome(Nome: String);
var
   i : integer;
begin
    First;
    indiceatual := 0;
    for i := 0 to Count-1 do
    begin
        if TPessoa_Fisica(Items[i]).Nome = Nome then
        begin
            indiceatual := i;
            Exit;
        end;
    end;
end;

procedure TFormListaObjetos.FormCreate(Sender: TObject);
begin
     ListaContribuinte := TListaContribuinte.Create;
end;

function TContribuinte.calcula_imposto:double;
begin
     if (FRendimento < 900.0) then Fimposto := 0.0;
     if (FRendimento > 1800) then Fimposto := 0.25*rendimento -315.0;
     if (FRendimento >= 900)and(FRendimento <= 1800) then Fimposto := 0.15*rendimento -135.0;
     result := Fimposto;
end;

procedure TContribuinte.SetRendimento (Valor : double);
begin
   if valor < 0
   then
     raise ERendimentoError.Create('Rendimento Não pode ser Negativo')
   else
     FRendimento := valor;
end;

procedure TFormListaObjetos.BotaoAdicionaClick(Sender: TObject);
var
   Contribuinte: TPessoa_Fisica;
begin
   Contribuinte := TPessoa_Fisica.Create;
   Contribuinte.Nome := EditNome.Text;
   Contribuinte.Rendimento := StrToFloat(EditRendimento.Text);
   ListaContribuinte.Add(Contribuinte);
   indiceatual := ListaContribuinte.Count-1;
   LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
   BotaoRemove.Enabled := True;
end;
```

```
procedure TFormListaObjetos.BotaoInsereClick(Sender: TObject);
var
   Contribuinte: TPessoa_Fisica;
begin
   Contribuinte := TPessoa_Fisica.Create;
   Contribuinte.Nome := EditNome.Text;
   Contribuinte. rendimento := StrToFloat(EditRendimento.Text);
   ListaContribuinte.Insert(indiceatual, Contribuinte);
   BotaoRemove.Enabled := True;
end;

procedure TFormListaObjetos.BotaoOrdenaClick(Sender: TObject);
begin
    ListaContribuinte.Sort(Compara);
end;

procedure TFormListaObjetos.BotaoPrimeiroClick(Sender: TObject);
begin
     ListaContribuinte.First;
     indiceatual := 0;
     EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
     EditRendimento.Text :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).rendimento);
     LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
end;

procedure TFormListaObjetos.BotaoAnteriorClick(Sender: TObject);
begin
     if indiceatual > 0 then indiceatual := indiceatual-1;
     EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
     EditRendimento.Text :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).rendimento);
     LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
end;

procedure TFormListaObjetos.BotaoProximoClick(Sender: TObject);
begin
     if indiceatual < ListaContribuinte.Count-1 then indiceatual := indiceatual+1;
     EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
     EditRendimento.Text :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).rendimento);
     LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
end;

procedure TFormListaObjetos.BotaoUltimoClick(Sender: TObject);
begin
     ListaContribuinte.Last;
     indiceatual := ListaContribuinte.Count-1;
     EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
     EditRendimento.Text :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).rendimento);
     LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
end;

procedure TFormListaObjetos.EditPesquisaChange(Sender: TObject);
begin
    ListaContribuinte.PesquisaNome(EditPesquisa.Text);
     EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
     EditRendimento.Text :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).rendimento);
```

```
     LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
end;

procedure TFormListaObjetos.EditNomeEnter(Sender: TObject);
begin
     EditNome.Clear;
     LabelValorImposto.Caption := '0.0';
end;

procedure TFormListaObjetos.EditRendimento Enter(Sender: TObject);
begin
     EditRendimento.Clear;
end;

procedure TFormListaObjetos.BotaoRemoveClick(Sender: TObject);
begin
     ListaContribuinte.Delete(indiceatual);
     if indiceatual > ListaContribuinte.Count-1 then indiceatual := ListaContribuinte.Count-1;
     if ListaContribuinte.Count > 0 then
     begin
          EditNome.Text := TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).Nome;
          EditRendimento.Text :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).rendimento);
          LabelValorImposto.Caption :=
FloatToStr(TPessoa_Fisica(ListaContribuinte.Items[indiceatual]).calcula_imposto);
     end
     else
     begin
          EditNome.Clear;
          EditRendimento.Clear;
          LabelValorImposto.Caption := '';
          BotaoRemove.Enabled := False;
     end;
end;

procedure TFormListaObjetos.FormDestroy(Sender: TObject);
begin
     ListaContribuinte.Free;
end;
end.
```

Evidentemente esse exemplo pode ser ainda mais incrementado pelo leitor, pois nosso objetivo aqui foi apresentar as técnicas fundamentais relacionadas à manipulação de listas de objetos.

Um dos problemas que deve ter sido verificado pelo leitor é a ausência de persistência de dados, ou seja, os dados são manipulados apenas em memória e perdidos após o término da aplicação.

KNOW-HOW EM: SOBRECARGA DE MÉTODOS

PRÉ-REQUISITOS

- Conhecimento da sintaxe básica da linguagem Pascal e dos fundamentos da programação orientada a objetos em Delphi 7.

METODOLOGIA

- Apresentação do conceito de sobrecarga de métodos.

O Conceito de Sobrecarga de Métodos

Conforme você já deve ter verificado, muitas classes têm diversos construtores, cada um com um nome distinto (um exemplo típico é o da classe Exception, já apresentada).

Isso decorre do fato de que, até a versão 3, o Delphi não permitia a utilização do recurso de sobrecarga de métodos (já existente há muito tempo no C++) – recurso este que permite que se definam vários métodos de mesmo nome, mas com uma lista de parâmetros distintos (em quantidade e/ou tipo).

A classe TContribuinte, por exemplo, poderia definir dois métodos construtores, recodificando a classe da seguinte maneira:

```
TContribuinte = Class
   private
      // Campos e métodos privados devem ser definidos aqui.
      FRendimento : double;
      Fimposto : double;
   public
      constructor Create;overload;
      constructor Create(rendimento_inicial : double);overload;
      procedure SetRendimento (valor : double);
      function calcula_imposto:double;
      property rendimento : double read FRendimento write SetRendimento;
      property imposto : double read calcula_imposto;
   protected
      // Campos e métodos protegidos devem ser definidos aqui.
   end;
```

As implementações dos métodos seriam feitas da seguinte maneira:

```
constructor TContribuinte.Create(rendimento_inicial: double);
begin
    inherited Create;
    rendimento := rendimento_inicial;
end;

constructor TContribuinte.Create;
begin
    inherited Create;
    rendimento := 0.0;
end;
```

Repare que, agora, a classe possui dois métodos de mesmo nome, mas com uma lista de parâmetros distinta.

A única exigência que deve ser atendida ao se sobrecarregar um método consiste em incluir na sua declaração a palavra reservada overload, conforme exemplificado anteriormente.

É importante destacar que esse recurso pode ser aplicado a qualquer método da classe, e não apenas aos seus construtores, e que os códigos dos métodos sobrecarregados podem ser inteiramente distintos.

Nesse caso, as duas linhas de código a seguir são igualmente válidas:

```
Contribuinte := TContribuinte.Create;
```

e

```
Contribuinte := TContribuinte.Create(1000.00);
```

A diferença é que, no primeiro caso, a instância da classe ao ser criada atribui o valor 0.0 ao campo rendimento, enquanto que, no segundo caso, esse campo recebe o valor passado como parâmetro.

O recurso de sobrecarga também pode ser empregado em funções e procedimentos que não sejam métodos de uma classe.

KNOW-HOW EM: DEFINIÇÃO DE PARÂMETROS DEFAULT PARA UMA FUNÇÃO OU PROCEDIMENTO

PRÉ-REQUISITOS

- Conhecimento da sintaxe básica da linguagem Pascal e dos fundamentos da programação orientada a objetos em Delphi 7.

METODOLOGIA

- Apresentação do conceito de definição de parâmetros default em uma função ou procedimento.

O CONCEITO DE PARÂMETRO DEFAULT PARA UMA FUNÇÃO OU PROCEDIMENTO

Outro recurso bastante interessante do Delphi 7 é o associado à definição de uma função ou procedimento com parâmetros default.

Esse recurso permite que aos últimos parâmetros da lista de parâmetros de uma função ou procedimento seja definido um valor default.

Dessa maneira, se um valor não for especificado para esses parâmetros na chamada da função ou procedimento, esses assumirão os seus valores default.

Considere, por exemplo, a seguinte função, destinada a retornar o valor da soma de três números inteiros:

Declaração:

```
function Soma(valor1, valor2, valor3 : integer):integer;
```

Implementação:

```
function Soma(valor1, valor2, valor3 : integer):integer;
begin
     result := valor1+valor2+valor3;
end;
```

Caso se queira definir o valor 0 como valor default para os dois últimos parâmetros, essa função deveria ser redeclarada da seguinte maneira (não altere a sua implementação):

```
function Soma(valor1 : integer; valor2 : integer = 0; valor3 : integer = 0):integer;
```

Dessa maneira, as chamadas da função a seguir são equivalentes:

```
Total := Soma(10,0,0);
Total := Soma(10,0);
Total := Soma(10);
```

Com os recursos de sobrecargas de métodos e de definição de valores default, o Object Pascal se torna uma linguagem de programação orientada a objetos ainda mais completa.

Evidentemente, esses recursos também se aplicam a métodos de classes.

Capítulo 20

O Conceito de Componentes

Ao utilizar o Delphi 7 como uma ferramenta de desenvolvimento de aplicações, trabalhamos intensamente com os componentes fornecidos por essa ferramenta.

Como já deve ser do seu conhecimento, o Delphi 7 fornece componentes capazes de realizar as mais diversas tarefas, como por exemplo:

- Criar elementos de interface como menus pull-down, menus pop-up, botões de comando, caixas de texto, etc.
- Representar caixas de diálogo padrão do Windows, com as quais nosso usuário já está habituado, em função da sua experiência prévia decorrente da utilização de outras aplicações.
- Acessar tabelas de bancos de dados, através dos métodos fornecidos pelos componentes e mediante a utilização de declarações SQL.
- Exibir os valores armazenados nos campos dos registros de uma tabela.
- Criar relatórios.
- Incorporar recursos de multimídia à sua aplicação.

Existem situações, no entanto, em que os componentes oferecidos pelo ambiente de desenvolvimento não são suficientes para atender a todas as nossas necessidades, embora sejam capazes de atendê-las parcialmente.

Nesses casos, têm-se a opção de instalar componentes desenvolvidos por terceiros ou criar novos componentes a partir de componentes já existentes, incorporando apenas as características necessárias a atender completamente às nossas necessidades.

O objetivo deste capítulo será, portanto, apresentar ao leitor o mecanismo de definição de componentes, de forma a torná-lo apto a estender as potencialidades oferecidas pelos componentes já disponibilizados pelo ambiente de desenvolvimento do Delphi 7.

A criação de novos componentes será objeto de um capítulo específico.

KNOW-HOW em: Definição de Componentes

PRÉ-REQUISITOS

- Conhecimento da sintaxe básica da linguagem Pascal e dos fundamentos da programação orientada a objetos em Delphi 7.

METODOLOGIA

- Apresentação do conceito de componentes.
- Definição dos conceitos de propriedades e eventos.
- Apresentação da classe TCanvas.

TÉCNICA

- Discussão dos conceitos relacionados à definição de um componente.
- Implementação de procedimentos capazes de responder a mensagens do sistema operacional.

O Conceito de Componentes

Os componentes são, na realidade, objetos ou instâncias de classes que têm algumas características adicionais às existentes nas classes definidas no capítulo anterior.

Diferentemente do que ocorre com as instâncias das classes definidas anteriormente, os componentes podem ter algumas das suas características manipuladas durante a fase de projeto de um aplicativo, isto é, a partir do próprio ambiente de desenvolvimento integrado do Delphi 7, e isso normalmente é feito alterando-se algumas das suas propriedades diretamente no Object Inspector.

O próprio componente pode ser visualizado e manipulado diretamente com o mouse (e algumas das suas características visuais, como tamanho e posição, podem ser diretamente alteradas pelo mouse).

Além disso, durante a execução do aplicativo, alguns componentes podem responder a ações diretas do usuário ou do sistema operacional, ou seja, são capazes de responder às mensagens disparadas pelo sistema operacional. O usuário do componente pode ainda definir um trecho de código a ser executado quando o componente responder a essa mensagem, o que é geralmente feito definindo-se procedimentos associados a eventos (onde se define genericamente por evento uma ação do usuário ou do sistema operacional).

Exemplificando: Quando um componente precisa ser redesenhado na tela, o sistema operacional envia ao componente uma mensagem identificada como WM_Paint.

Além disso, alguns componentes possuem um evento chamado OnPaint, cujo procedimento associado será executado sempre que o componente precisar responder a essa mensagem.

Alguns componentes, no entanto, só podem ser visualizados na fase de projeto do aplicativo, como por exemplo aqueles que representam as caixas de diálogo padrões do Windows. Esse tipo de componente, na realidade, representa uma determinada funcionalidade do sistema operacional, que foi transformada em componente para facilitar o trabalho do programador. O mesmo ocorre com os componentes utilizados para representar tabelas, em aplicações que acessam bancos de dados.

Se você examinar o diagrama da hierarquia de objetos do Delphi 7, verá que todas as classes usadas na definição de componentes são classes derivadas, direta ou indiretamente, da classe TPersistent, definida na unit Classes.

A classe TPersistent incorpora a funcionalidade básica a todas as classes cujas instâncias devem ser capazes de ler e armazenar suas características em um meio auxiliar de armazenamento, que pode ser a memória RAM ou um disco magnético. Na realidade, os métodos fornecidos por essa classe são todos virtuais ou dinâmicos, podendo portanto ser sobrecarregados nas classes dela derivadas por herança, de forma a ampliar a sua funcionalidade.

Além dos componentes propriamente ditos, todas as classes cujos objetos precisam incorporar o conceito de persistência são derivadas diretamente da classe TPersistent.

Dentre as classes derivadas diretamente da classe TPersistent, podem-se destacar:

- TCanvas: Utilizada para representar uma superfície de desenho, incorporando boa parte da funcionalidade presente na GDI (Graphical Device Interface) do Windows.

- TClipboard: Utilizada para representar a área de transferência do Windows.
- TStrings: Definida como classe-base de muitas outras classes destinadas à manipulação de listas de strings.

A classe TComponent, derivada diretamente da classe TPersistent, também é uma classe-base de todos os componentes definidos na linguagem Object Pascal, e implementa algumas características e funcionalidades comuns a todos os componentes, como por exemplo:

- Exibir uma imagem na paleta de componentes do ambiente de desenvolvimento integrado do Delphi 7.
- Exibir o componente em um formulário.
- Incorporar e gerenciar outros componentes. Nesse caso, diz-se que esse componente é o proprietário dos componentes nele inseridos.

Dessa maneira, todos os componentes são derivados diretamente ou indiretamente da classe TComponent. Dentre as classes derivadas diretamente de TComponent, podem-se destacar:

- TApplication: Essa classe permite a definição de instâncias que representam uma aplicação.
- TBatchMove: Os objetos dessa classe são usados para efetuar transferência de registros entre tabelas.
- TCommomDialog: Usada como classe-base de todas as classes usadas na representação das caixas de diálogo padrões do Windows.
- TDataSet: Usada como classe-base para todas as classes destinadas a acessar as tabelas de um banco de dados.
- TMenu: Usada para definir o menu principal de uma aplicação.

Uma característica comum a essas classes é o fato de que os componentes por elas representados não são visíveis durante a execução do aplicativo. O componente SQLDataset (derivado da classe TDataset), por exemplo, pode ser manipulado durante a fase de projeto de uma aplicação, mas não é exibido durante a execução do aplicativo.

Analisando-se novamente o diagrama de hierarquia de objetos do Delphi 7, pode-se verificar que todos os componentes que apresentam esse tipo de componente são derivados de TComponent, mas não possuem a classe TControl como uma das suas classes-base.

Dessa maneira, pode-se concluir: "Componentes cujas instâncias não devem ser visualizadas durante a execução do aplicativo devem ser representados por classes derivadas direta ou indiretamente da classe TComponent, mas não devem ser derivadas da classe TControl."

Por último, pode-se verificar a partir do diagrama de hierarquia de objetos do Delphi 7 que existe ainda uma classe chamada TControl, derivada de TComponent, da qual são derivadas muitas outras classes que representam os principais componentes de construção da interface.

A principal característica dessa classe reside na incorporação da funcionalidade necessária para que os objetos das classes dela derivadas por herança possam ser visualizados durante a execução do aplicativo.

Dessa maneira, pode-se concluir: "Componentes cujas instâncias devem ser visualizadas durante a execução do aplicativo devem ser representados por classes derivadas da classe TControl."

A classe TControl, por sua vez, possui duas subclasses principais, denominadas TGraphicControl e TwinControl.

A classe TGraphicControl é utilizada como classe-base para as classes que representam componentes que devem exibir texto e gráfico, mas que não devem receber o foco da aplicação durante a sua execução, isto é, componentes que não são selecionados quando se pressiona seguidamente a tecla Tab, nem podem conter outros componentes.

A classe TwinControl (definida como sinônimo da classe TWidget na CLX), por sua vez, é utilizada como classe-base para as classes que representam componentes que devem ter as seguintes características:

- São capazes de receber o foco da aplicação durante a sua execução.
- Podem conter outros controles.
- Possuem um handle que os identifica.

Exibindo uma Propriedade no Object Inspector

No capítulo anterior foi apresentado o conceito de propriedade, definida como sendo um meio de comunicação entre um campo interno e privado da classe e o código que utiliza um objeto dessa classe.

Normalmente, para que uma propriedade possa ser acessada pelo código da aplicação, esta deve ser definida na seção public de uma classe.

No caso de um componente, para que uma propriedade possa ter o seu valor visualizado e alterado no Object Inspector, você deve defini-la na seção published da classe que representa o componente, seção esta que não havia sido abordada até o momento, e será visto em maiores detalhes no capítulo referente à criação de componentes.

O Sistema de Mensagens do Windows

Conforme descrito anteriormente, um componente deve ser capaz de responder a determinadas mensagens do Windows, o que normalmente é feito definindo-se procedimentos associados a um evento que tenha sido configurado para responder a essa mensagem.

O Delphi 7 define uma série de constantes numéricas, relacionadas na unit messages.pas, que identificam as diversas mensagens que podem ser enviadas a um componente pelo sistema operacional.

A Mensagem WM_Paint do Windows, por exemplo, é codificada da seguinte maneira pela VCL do Delphi 7:

```
WM_PAINT = $000F;
```

As principais mensagens relacionadas com o pressionamento de teclas e movimentação do mouse para aplicações desenvolvidas em Delphi 7 com a VCL são codificadas como mostrado a seguir:

```
WM_MOUSEFIRST = $0200;
WM_MOUSEMOVE = $0200;
WM_LBUTTONDOWN = $0201;
```

```
WM_LBUTTONUP = $0202;
WM_LBUTTONDBLCLK = $0203;
WM_RBUTTONDOWN = $0204;
WM_RBUTTONUP = $0205;
WM_RBUTTONDBLCLK = $0206;
WM_MBUTTONDOWN = $0207;
WM_MBUTTONUP = $0208;
WM_MBUTTONDBLCLK = $0209;
WM_MOUSEWHEEL = $020A;
WM_MOUSELAST = $020A;
```

No caso do Windows, para definir em uma classe um método que responda a uma mensagem do sistema operacional, basta codificá-lo com a seguinte sintaxe:

```
Procedure <código_mensagem_sem_undescore>(var Nome_Parâmetro: TMessage);message
<código_da_mensagem>;
```

Onde:

- < código_mensagem_sem_undescore> é exatamente isso: o código da mensagem sem o undescore. Conseqüentemente, se quisermos definir um método que responda à mensagem WM_PAINT, ele será denominado WMPaint.
- Nome_Parâmetro é o nome de um parâmetro passado por referência, e de um tipo definido como um registro que identifica a mensagem a ser manipulada, e definida na unit Messages.
- Message é uma palavra reservada utilizada para identificar que esse procedimento será executado em resposta a uma mensagem do sistema operacional.
- <código_da_mensagem> é o código da mensagem propriamente dita.

Para responder a uma mensagem WM_PAINT, por exemplo, um componente deve definir um método chamado WMPaint, que apresenta o seguinte cabeçalho:

```
procedure WMPaint(var Message: TWMPaint); message WM_PAINT;
```

As classes TWinControl e TGraphicControl, por exemplo, implementam esse método, embora o façam de maneira diferente, como mostram os trechos de código que implementam esses métodos, reproduzidos a seguir.

```
procedure TWinControl.WMPaint(var Message: TWMPaint);
var
  DC, MemDC: HDC;
  MemBitmap, OldBitmap: HBITMAP;
  PS: TPaintStruct;
begin
  if not FDoubleBuffered or (Message.DC <> 0) then
    if ControlCount = 0 then inherited else PaintHandler(Message)
  else
  begin
    DC:= GetDC(0);
    MemBitmap:= CreateCompatibleBitmap(DC, ClientRect.Right, ClientRect.Bottom);
    ReleaseDC(0, DC);
    MemDC:= CreateCompatibleDC(0);
    OldBitmap:= SelectObject(MemDC, MemBitmap);
    try
      DC:= BeginPaint(Handle, PS);
      Perform(WM_ERASEBKGND, MemDC, MemDC);
```

```
      Message.DC:= MemDC;
      WMPaint(Message);
      Message.DC:= 0;
      BitBlt(DC, 0, 0, ClientRect.Right, ClientRect.Bottom, MemDC, 0, 0, SRCCOPY);
      EndPaint(Handle, PS);
    finally
      SelectObject(MemDC, OldBitmap);
      DeleteDC(MemDC);
      DeleteObject(MemBitmap);
    end;
  end;
end;

procedure TGraphicControl.WMPaint(var Message: TWMPaint);
begin
  if Message.DC <> 0 then
  begin
    Canvas.Lock;
    try
      Canvas.Handle:= Message.DC;
      try
        Paint;
      finally
        Canvas.Handle:= 0;
      end;
    finally
      Canvas.Unlock;
    end;
  end;
end;
```

Alguns exemplos de tipos definidos como registro (tipo composto) na unit Messages são apresentados a seguir.

Para representar mensagens geradas pelo pressionamento de uma tecla:

```
TWMKey = record
    Msg: Cardinal;
    CharCode: Word;
    Unused: Word;
    KeyData: Longint;
    Result: Longint;
  end;
```

Para representar mensagens geradas pelo mouse:

```
  TWMMouse = record
    Msg: Cardinal;
    Keys: Longint;
    case Integer of
      0: (
        XPos: Smallint;
        YPos: Smallint);
      1: (
        Pos: TSmallPoint;
        Result: Longint);
  end;
```

Para representar a mensagem WM_Paint:

```
TWMPaint = record
    Msg: Cardinal;
```

```
    DC: HDC;
    Unused: Longint;
    Result: Longint;
  end;
```

Algumas mensagens específicas são representadas como um alias (nome alternativo) para os registros descritos anteriormente, como exemplificado a seguir:

```
TWMLButtonDblClk = TWMMouse;
TWMLButtonDown   = TWMMouse;
TWMLButtonUp     = TWMMouse;
TWMMButtonDblClk = TWMMouse;
TWMMButtonDown   = TWMMouse;
TWMMButtonUp     = TWMMouse;
TWMMouseMove = TWMMouse;
TWMRButtonDblClk = TWMMouse;
TWMRButtonDown = TWMMouse;
TWMRButtonUp = TWMMouse;
TWMChar = TWMKey;
TWMKeyDown = TWMKey;
TWMKeyUp = TWMKey;
TWMSysChar = TWMKey;
TWMSysKeyDown = TWMKey;
TWMSysKeyUp = TWMKey;
```

Não discutiremos neste livro a codificação desses métodos, que executam diversas funções da API do Windows. O objetivo desse tópico é mostrar como são implementados os métodos que respondem a mensagens do sistema operacional.

Repare que, como a classe TGraphicControl já implementa o método Paint, tenta executá-lo a partir do procedimento definido como resposta à mensagem WM_PAINT, o que não ocorre no caso do mesmo método para a classe TWinControl, que não implementa o método Paint.

Entretanto, a classe TCustomControl, derivada da classe TWinControl e que implementa o método Paint, define seu procedimento WMPaint da maneira descrita a seguir:

```
procedure TCustomControl.WMPaint(var Message: TWMPaint);
begin
  PaintHandler(Message);
end;
```

Esse método executa o método PaintHandler, definido na classe-base, e que por sua vez executa o método PaintWindow, que na classe TCustomControl é implementado da seguinte forma:

```
procedure TCustomControl.PaintWindow(DC: HDC);
begin
  FCanvas.Lock;
  try
    FCanvas.Handle:= DC;
    try
      Paint;
    finally
      FCanvas.Handle:= 0;
    end;
  finally
    FCanvas.Unlock;
  end;
end;
```

Repare que esse método chama o método Paint.

O importante neste ponto é compreender que:

- Para responder à mensagem WM_PAINT do Windows, uma classe deve implementar o método WMPaint da maneira descrita anteriormente.
- Se uma classe implementa o método Paint, este deve ser chamado a partir do seu método WMPaint, ou por qualquer método chamado a partir deste.

A Classe TCanvas

Muitos componentes (como os representados pela classe TGraphicControl ou de classes dela derivadas por herança) têm um campo interno chamado FCanvas, e uma propriedade chamada Canvas, da classe TCanvas, que representa a superfície de desenho do componente.

Neste tópico apresentaremos algumas propriedades e métodos da classe TCanvas, fundamentais à compreensão dos procedimentos envolvidos na criação de componentes.

Para o Windows, qualquer área a ser manipulada na tela (ou na impressora) é tratada como uma superfície de desenho.

Para simplificar a vida do programador, o Delphi 7 possui uma classe, denominada TCanvas, que representa uma superfície de desenho retangular sobre a qual poderão ser feitos desenhos, exibidos textos, etc.

Em um Canvas existe um sistema de coordenadas cartesianas em que a origem está situada no canto superior esquerdo, as abscissas crescem da esquerda para a direita e as ordenadas de cima para baixo (ao contrário do sistema cartesiano convencional).

Conforme será visto posteriormente, no capítulo referente à programação gráfica, existem funções da GDI do Windows que permitem que se altere a orientação de eixos do Canvas, bem como o posicionamento da origem desse sistema.

A figura a seguir apresenta a orientação de eixos em um Canvas, conforme descrito anteriormente.

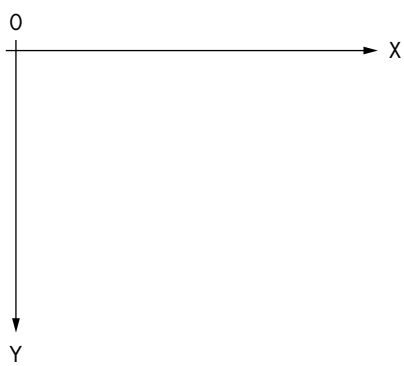


Figura 20.1: Sistema de coordenadas da tela.

Guarde bem este conceito: um canvas é um objeto da classe TCanvas, que define uma região retangular da tela, e que possui, entre seus métodos, as principais funções da GDI do Windows.

O desenho de linhas em um Canvas é feito usando uma caneta imaginária, que é na realidade um objeto da classe TPen (outra classe definida pelo Delphi 7), e o preenchimento de superfícies é feito usando-se um pincel imaginário, que é na realidade um objeto da classe TBrush. Dentre as principais propriedades de um objeto da classe TPen – a caneta imaginária – destacam-se aquelas que definem a sua espessura, a sua cor, modo e estilo de desenho. Para um objeto da classe TBrush, suas principais propriedades são aquelas que definem a sua cor e estilo.

Principais Propriedades da Classe TCanvas

Dentre as principais propriedades da classe TCanvas, destacam-se:

Brush

A propriedade Brush de um objeto da classe TCanvas é, na realidade, um objeto da classe TBrush, e define as características do pincel utilizado no preenchimento de superfícies como retângulos e elipses, por exemplo.

CanvasOrientation

Essa propriedade, apenas de leitura, é uma variável do tipo Tcanvas e pode assumir um dos seguintes valores:

- coLeftToRight: Orientação da esquerda para a direita.
- coRightToLeft: Orientação da direita para a esquerda.

ClipRect

Essa propriedade é um objeto da classe TRect, e permite restringir a área de desenho a uma porção retangular do Canvas.

CopyMode

Essa propriedade é definida como uma variável inteira, e define como imagens gráficas devem ser desenhadas no Canvas: Uma relação dos valores possíveis e seus efeitos será apresentada no capítulo de programação gráfica.

Font

Essa propriedade é um objeto da classe TFont, e define a fonte utilizada na exibição de textos no Canvas.

Handle

Essa propriedade retorna o handle do Canvas, e é utilizada apenas quando se precisa utilizar funções gráficas da GDI do Windows que não foram incorporadas pela VCL do Delphi 7 ou pela CLX como métodos da classe TCanvas.

LockCount

Essa propriedade é definida como uma variável inteira, e define quantas vezes o canvas foi protegido contra a intervenção de outras threads, mediante chamadas ao seu método Lock.

Pen

A propriedade Pen de um objeto da classe TCanvas é, na realidade, um objeto da classe TPen, e define as características da caneta utilizada no desenho de linhas e do contorno de figuras como retângulos e elipses, por exemplo.

PenPos

Essa propriedade é uma variável do tipo TPoint, e define a posição atual da caneta no Canvas. Atribuir um valor à propriedade PenPos equivale a executar o método MoveTo do Canvas.

Pixels [x,y]

Essa propriedade define a cor do pixel situado nas coordenadas (x,y) do Canvas.

TextFlags

Essa propriedade é definida como uma variável inteira, e define como um texto deve ser desenhado no Canvas. Uma relação dos valores possíveis e seus efeitos será apresentada no capítulo de programação gráfica.

Principais Métodos da Classe TCanvas

Dentre os principais métodos da classe TCanvas, destacam-se:

Arc

Declaração

```
procedure Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
```

Esse método desenha na superfície do Canvas um arco coincidente com o perímetro da elipse cujo retângulo circunscrito é definido pelos pontos (X1,Y1) e (X2,Y2). O arco começa no ponto de interseção desse retângulo com a linha que une o centro da elipse ao ponto (X3,Y3) e segue no sentido anti-horário até o ponto de interseção desse retângulo com a linha que une o centro da elipse ao ponto (X4,Y4).

BrushCopy

Declaração

```
procedure BrushCopy(const Dest: TRect; Bitmap: TBitmap; const Source: TRect; Color: TColor);
```

Esse método copia uma porção retangular de um bitmap (Source) para uma área retangular do canvas (Dest), substituindo uma das cores do bitmap (Color) pela definida para a propriedade Brush do canvas.

Chord

Declaração

```
procedure Chord(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
```

Esse método desenha uma curva fechada, definida pela interseção entre uma linha e uma elipse. A Elipse é definida pelos dois primeiros pontos e a linha, pelos dois últimos.

CopyRect

Declaração

```
procedure CopyRect(Dest: TRect; Canvas: TCanvas; Source: TRect);
```

Esse método copia uma área retangular de um Canvas para outra área retangular do Canvas corrente (que executa o método).

Draw

Declaração

```
procedure Draw(X, Y: Integer; Graphic: TGraphic);
```

Esse método copia uma imagem gráfica (que pode ser um ícone, um bitmap ou um metafile) para o Canvas corrente, sendo a extremidade superior esquerda da imagem colocada no ponto de coordenadas X e Y (passadas como parâmetros).

DrawFocusRect

Declaração

```
procedure DrawFocusRect(const Rect: TRect);
```

Esse método desenha um retângulo no Canvas, com o estilo comumente usado para indicar que o componente recebeu o foco da aplicação.

Ellipse

Declaração

```
procedure Ellipse(X1, Y1, X2, Y2: Integer); overload;
procedure Ellipse(const Rect: TRect); overload;
```

Esse método desenha no Canvas uma elipse cujo retângulo circunscrito é definido pelos pontos (X1,Y1) e (X2,Y2) ou por uma área retângular (Rect). Para desenhar um círculo, basta que o retângulo circunscrito seja um quadrado, isto é, (Y2 - Y1) deve ser igual a (X2 - X1). Repare que este método é um método sobrecarregado.

FillRect

Declaração

```
procedure DrawFocusRect(const Rect: TRect);
```

Esse método preenche uma área retangular do Canvas (passada como parâmetro) com o pincel (brush) corrente.

LineTo

Declaração

```
procedure LineTo(X, Y: Integer);
```

Esse método desenha no Canvas uma linha que começa na posição atual da caneta imaginária e termina na posição definida pelas coordenadas (X,Y), que passa a ser a posição atual da caneta imaginária.

Lock

Declaração

```
procedure Lock;
```

Esse método impede que outras threads desenhem sobre o Canvas.

MoveTo

Declaração

```
procedure MoveTo(X, Y: Integer);
```

Esse método move a caneta imaginária para a posição do Canvas definida pelas coordenadas (X,Y).

PolyBezier

Declaração

```
procedure PolyBezier(const Points: array of TPoint);
```

Esse método desenha uma curva de Bezier, com base no conjunto de pontos fornecido como parâmetro.

Pie (X1, Y1, X2, Y2)

Declaração

```
procedure Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Longint);
```

Esse método desenha no Canvas um setor elíptico cujo retângulo circunscrito é definido pelos pontos (X1,Y1) e (X2,Y2), preenchendo-o com o pincel definido na propriedade Brush do Canvas.

Polygon

Declaração

```
procedure Polygon(Points: array of TPoint);
```

Esse método desenha no Canvas um polígono fechado usando a caneta imaginária definida na propriedade Pen do Canvas e preenchendo-o com o pincel imaginário definido pela propriedade Brush do Canvas.

Polyline

Declaração

```
procedure Polyline(Points: array of TPoint);
```

Esse método desenha uma poligonal aberta unindo os pontos que compõem o array passado como parâmetro. Para desenhar uma poligonal fechada, basta que as coordenadas do primeiro e do último ponto do array sejam coincidentes.

Rectangle

Declaração

```
procedure Rectangle(X1, Y1, X2, Y2: Integer); overload;
procedure Rectangle(const Rect: TRect); overload;
```

Esse método desenha no Canvas um retângulo definido pelos pontos (X1,Y1) e (X2,Y2), ou por uma variável do tipo Trect (repare que este é um método sobrecarregado) preenchendo-o com o pincel definido na propriedade Brush do Canvas.

RoundRect

Declaração

```
procedure RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer);
```

Esse método desenha no Canvas um retângulo de cantos arredondados definido pelos pontos (X1,Y1) e (X2,Y2), sendo os cantos concordados por uma elipse de eixos iguais a X3 e Y3.

StretchDraw

Declaração

```
procedure StretchDraw(const Rect: TRect; Graphic: TGraphic );
```

Esse método copia, na área retangular definida pelo parâmetro Rect do canvas corrente, uma imagem definida pelo parâmetro Graphic.

TextExtent

Declaração

```
function TextExtent(const Text: string): TSize;
```

Esse método retorna, nos campos cx e cy de uma variável do tipo TSize, as dimensões (largura e altura, em pixels) que uma string ocupa ao ser desenhada no canvas corrente.

TextHeight

Declaração

```
function TextHeight(const Text: string): Integer;
```

Esse método retorna a altura, em pixels, que a string passada como parâmetro ocupa no Canvas.

TextOut

Declaração

```
procedure TextOut(X, Y: Integer; const Text: string);
```

Esse método exibe no Canvas uma string definida pelo parâmetro Texto, na posição definida pelas coordenadas (X,Y). Essas coordenadas definem o vértice superior esquerdo do retângulo circunscrito ao texto, usando a fonte corrente do Canvas (definida pela sua propriedade Font).

TextWidth

Declaração

```
function TextWidth(const Text: string): Integer;
```

Esse método retorna a largura, em pixels, que a string definida pelo parâmetro Texto ocupa no Canvas.

Unlock

Declaração

```
procedure Unlock;
```

Esse método permite que outras threads desenhem sobre o Canvas. Deve ser chamado sempre que outras threads houvessem sido impedidas de desenhar no Canvas após uma chamada do método Lock.

Apresentam-se a seguir as principais propriedades da classe TPen.

Principais Propriedades da Classe TPen

Dentre as principais propriedades da classe TPen, destacam-se:

Color

Essa propriedade é um objeto da classe TColor, e define a cor atual da caneta imaginária. Você pode usar uma das constantes predefinidas do Delphi 7 (clBlack, clWhite, clBlue, clYellow, etc.).

Style

Essa propriedade define como a linha será desenhada, e pode assumir um dos valores apresentados a seguir.

Valor	Significado
psSolid	Desenha uma linha sólida.
psDash	Desenha uma linha tracejada.
psDot	Desenha uma linha pontilhada.

continua

Valor	Significado
psDashDot	Desenha uma linha no estilo traço-ponto.
psDashDotDot	Desenha uma linha no estilo traço-dois pontos.
psClear	Desenha uma linha invisível.
psInsideFrame	Desenha uma linha sólida, mas cuja cor pode sofrer leves variações, quando sua propriedade width tiver valor maior do que 1.

Mode

Essa propriedade define como a cor atual da caneta interage com a cor corrente do Canvas, e pode assumir um dos valores apresentados a seguir:

- pmBlack: A linha é desenhada na cor preta, independente da cor existente no Canvas.
- pmWhite: A linha é desenhada na cor branca, independente da cor existente no Canvas.
- pmNop: A cor atual do canvas permanece inalterada. A caneta se comporta de forma transparente.
- pmNot: A cor utilizada pela caneta será a inversa da presente no Canvas.
- pmCopy: A caneta desenha linhas com a cor especificada na sua propriedade Color.
- pmNotCopy: A caneta desenha linhas com a cor especificada na sua propriedade Color.
- pmMergePenNot: A caneta desenha linhas com a cor resultante da combinação entre a cor da propriedade Color da caneta e a inversa da presente no Canvas.
- pmMaskPenNot: A caneta desenha linhas com a cor resultante da combinação das cores comuns entre a cor da propriedade Color da caneta e a inversa da presente no Canvas.
- pmMergeNotPen: A caneta desenha linhas com a cor resultante da combinação entre a cor inversa da propriedade Color da caneta e a cor presente no Canvas.
- pmMaskNotPen: A caneta desenha linhas com a cor resultante da combinação entre a cor inversa da definida pela propriedade Color da caneta e a da cor de fundo do Canvas.
- pmMerge: A caneta desenha linhas com a cor resultante da combinação entre a cor definida pela propriedade Color da caneta e a da cor de fundo do Canvas.
- pmNotMerge: Cor inversa da definida com pmMerge.
- pmMask: A caneta desenha linhas com a cor resultante da combinação entre a cor definida pela propriedade Color da caneta e a da cor de fundo do Canvas.
- PmNotMask: Cor inversa da definida com pmMask.
- PmXor: A caneta desenha linhas com a cor resultante da combinação entre as cores presentes na propriedade Color da caneta e na cor de fundo do Canvas, mas não em ambas.
- PmNotXor: Cor inversa da definida com pmXor.

Width

Essa propriedade armazena um valor inteiro que define a espessura da linha desenhada pela caneta imaginária.

Apresentam-se a seguir as principais propriedades da classe TBrush.

Principais Propriedades da Classe TBrush

Dentre as principais propriedades da classe TBrush, destacam-se:

Bitmap

Essa propriedade permite que se defina um bitmap de 8 x 8 pixels como padrão de preenchimento do pincel imaginário. Se for atribuído um bitmap de dimensões maiores, apenas a região de 8 x 8 pixels situada no canto superior esquerdo do bitmap será considerada.

Color

Essa propriedade é um objeto da classe TColor, e define a cor atual do pincel imaginário.

Style

Essa propriedade define o estilo de preenchimento do pincel imaginário, com os seguintes valores:

TValor	Significado
bsSolid	Preenchimento Total (Sólido)
bsClear	Transparente
bsBDiagonal	Hachura em diagonal, com inclinação ////
bsFDiagonal	Hachura em diagonal, com inclinação \\\\
bsCross	Hachuras Ortogonais cruzadas
bsDiagCross	Hachuras Diagonais cruzadas
bsHorizontal	Hachuras Horizontais
bsVertical	Hachuras Verticais

O Componente Shape

O componente Shape, situado na página Additional da paleta de componentes, é utilizado para desenhar formas geométricas em um formulário ou outro componente que permita a inclusão de outros componentes em seu interior (como um Panel, por exemplo), e, diferentemente de outros componentes, permite que suas propriedades Brush e Pen sejam definidas durante o projeto da aplicação alterando os seus valores diretamente no Object Inspector.

A título de ilustração, apresentamos a seguir a definição da classe TShape e a implementação do seu método Paint, extraídos do arquivo ExtCtrls.pas:

```
TShape = class(TGraphicControl)
  private
  FPen: TPen;
    FBrush: TBrush;
    FShape: TShapeType;
    procedure SetBrush(Value: TBrush);
    procedure SetPen(Value: TPen);
    procedure SetShape(Value: TShapeType);
  protected
    procedure Paint; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    procedure StyleChanged(Sender: TObject);
    property Align;
    property Brush: TBrush read FBrush write SetBrush;
    property DragCursor;
    property DragMode;
    property Enabled;
    property ParentShowHint;
    property Pen: TPen read FPen write SetPen;
    property Shape: TShapeType read FShape write SetShape default stRectangle;
    property ShowHint;
    property Visible;
    property OnDragDrop;
    property OnDragOver;
    property OnEndDrag;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
    property OnStartDrag;
  end;
```

Observe que essa classe herda o campo interno FCanvas e a propriedade Canvas da sua classe-base, TGraphicControl.

```
procedure TShape.Paint;
var
  X, Y, W, H, S: Integer;
begin
  with Canvas do
  begin
    Pen := FPen;
    Brush := FBrush;
    X := Pen.Width div 2;
    Y := X;
    W := Width - Pen.Width + 1;
    H := Height - Pen.Width + 1;
    if Pen.Width = 0 then
    begin
      Dec(W);
      Dec(H);
    end;
    if W < H then S := W else S := H;
    if FShape in [stSquare, stRoundSquare, stCircle] then
    begin
      Inc(X, (W - S) div 2);
      Inc(Y, (H - S) div 2);
      W := S;
      H := S;
    end;
```

```
      case FShape of
        stRectangle, stSquare:
          Rectangle(X, Y, X + W, Y + H);
        stRoundRect, stRoundSquare:
          RoundRect(X, Y, X + W, Y + H, S div 4, S div 4);
        stCircle, stEllipse:
          Ellipse(X, Y, X + W, Y + H);
      end;
    end;
  end;
```

Conforme descrito anteriormente, o método Paint de um componente é executado sempre que o mesmo precisa ser redesenhado em um formulário. Repare que, no método Paint do componente Shape, são feitas várias chamadas a métodos da sua propriedade Canvas (que é um objeto da classe TCanvas).

Repare ainda que no método Paint do componente Shape, os valores das propriedades Brush e Pen do Canvas são definidos como sendo iguais aos definidos nos campos FBrush e FPen, que são na realidade objetos da classe TBrush e TPen, e que armazenam internamente os valores definidos pelas propriedades Brush e Pen, respectivamente.

Existem componentes, no entanto, que não permitem que se acessem suas propriedades Brush e Pen na fase de projeto, mas apenas durante a execução do aplicativo.

Esses componentes, no entanto, como aqueles representados pelas classes TPaintBox, TImage e TPrinter, possuem um evento chamado OnPaint, para o qual podemos definir um procedimento associado, e nesse procedimento definir o código referente a qualquer desenho a ser feito no seu Canvas.

Para que um desenho seja permanentemente visualizado em um formulário ou componente, o código que o define deve ser digitado no procedimento associado ao evento OnPaint do formulário ou componente. Se esse código não for incluído no evento OnPaint e a região do formulário ou componente que exibe o desenho for sobreposta por outra janela, este não será restaurado na tela quando a região correspondente se tornar visível novamente.

Capítulo 21

Criação de Componentes

O amplo conjunto de usuários do Delphi pode ser subdividido em dois grandes grupos: um grupo de desenvolvedores de aplicações e um grupo de desenvolvedores de componentes e ferramentas, que têm por objetivo ampliar as características e funcionalidades já disponíveis no ambiente de desenvolvimento do Borland Delphi.

Neste capítulo serão apresentados os procedimentos necessários à criação de componentes com o Borland Delphi, o que pode ser feito sem que seja necessária a utilização de qualquer ferramenta adicional, pois os novos componentes podem ser criados usando-se a própria linguagem Object Pascal.

Além disso, conforme será descrito posteriormente, os componentes desenvolvidos em Object Pascal podem ser transformados em controles ActiveX e utilizados em outros ambientes de desenvolvimento que suportam essa tecnologia.

Conforme será mostrado nos tópicos subseqüentes, você pode criar um componente inteiramente original ou partir de um componente já existente, usando o conceito de herança da linguagem Object Pascal. Nesse caso, você poderá adicionar novas características ao componente ancestral ou redefinir algumas de suas funcionalidades.

KNOW-HOW em: Criação de Componentes

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 5.
- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi.

METODOLOGIA

- Apresentação do problema: Utilização dos conceitos de herança e polimorfismo na criação de novos componentes.

TÉCNICA

- Apresentação dos procedimentos necessários à criação de novos componentes.

Apresentação do Problema

Ao longo dos últimos capítulos, você trabalhou intensamente com componentes, selecionando-os nas diversas páginas da paleta de componentes, inserindo-os em formulários e alterando algumas das suas propriedades diretamente no Object Inspector.

Esses componentes são, na verdade, objetos de classes previamente definidas na Visual Component Library – VCL (a biblioteca de componentes do Delphi). O que diferencia um componente de uma classe comum é justamente o fato de que os componentes podem ser manipulados na fase de projeto do aplicativo, ao passo que as classes comuns só podem ser manipuladas via código.

Existem situações, no entanto, em que precisamos adaptar um componente já existente de forma a adequá-lo às nossas necessidades.

Considere, por exemplo, que nossas aplicações utilizem constantemente caixas de texto (componentes da classe TEdit) nas quais devem ser digitados apenas números inteiros.

Existe uma solução natural, e que consiste em codificar adequadamente os procedimentos associados aos principais eventos desse componente.

Poderíamos, por exemplo, inserir em um formulário um componente caixa de texto e definir da seguinte maneira os procedimentos associados aos seus principais eventos:

```
procedure TFormNumerico.EditNumeroInteiroChange(Sender: TObject);
var
   indice : word;
   posinal : string;
begin
     indice := Pos('-',TEdit(Sender).text);
     if indice > 1
     then
        begin
             posinal := TEdit(Sender).Text;
          Delete(posinal, indice, 1);
             TEdit(Sender).Text := '-'+posinal;
        end;
     indice := Pos('+',TEdit(Sender).text);
     if indice > 1
     then
        begin
             posinal := TEdit(Sender).Text;
          Delete(posinal, indice, 1);
             TEdit(Sender).Text := '+'+posinal;
     end;
end;

procedure TFormNumerico.EditNumeroInteiroKeyPress(Sender: TObject;
  var Key: Char);
begin
if (not (key in ['0'..'9','-','+',#8]))or((key in ['+','-']) and
(Pos(key,TEdit(Sender).text)>0))
     then key := #0;
end;
```

O procedimento associado ao evento OnKeyPress do componente verifica se a tecla pressionada pelo usuário corresponde a um dos caracteres válidos, que são os dígitos (0 a 9) e os sinais "+" e "-".

Como a tecla pressionada é passada para o procedimento associado ao evento pelo parâmetro Key, basta verificar se a tecla representada por esse parâmetro pertence ao conjunto de valores permitidos, o que é feito nas linhas de código reproduzidas a seguir.

```
if (not (key in ['0'..'9','-','+',#8]))or((key in ['+','-']) and
(Pos(key,TEdit(Sender).text)>0))
then key := #0;
```

Essa linha de código verifica se a tecla representada pelo parâmetro Key pertence ao conjunto de valores permitidos e, caso essa condição não seja satisfeita, anula a entrada fornecida pelo usuário. Portanto, caso a tecla seja inválida, atribui-se o valor #0 ao parâmetro Key, anulando-se a ação do usuário (isso é possível, pois o parâmetro Key é passado por referência, como indica o modificador var incluído na relação de parâmetros do procedimento).

Repare que, no caso das teclas "+" e "-", deve-se ainda verificar se o caractere correspondente já existe na propriedade Text do componente, o que é feito verificando-se o valor retornado pela função Pos (que retorna a posição de um conjunto de caracteres em uma string) no procedimento associado ao

evento OnChange do componente. Se essa função retornar um valor positivo, indica que o sinal já está incluído na string representada pela propriedade Text do componente.

O procedimento associado ao evento OnChange do componente verifica se existe um caractere "+"ou "-" na sua propriedade Text, e, em caso positivo, verifica se a mesma foi inserida no meio da string. Em caso positivo, a remove da posição corrente da string e a recoloca no seu início, o que é feito no seguinte trecho de código:

Para o caractere "-":

```
indice := Pos('-',TEdit(Sender).text);
if indice > 1
then
   begin
        posinal := TEdit(Sender).Text;
     Delete(posinal, indice, 1);
        TEdit(Sender).Text := '-'+posinal;
   end;
```

Para o caractere "+":

```
indice := Pos('+',TEdit(Sender).text);
if indice > 1
then
   begin
        posinal := TEdit(Sender).Text;
 Delete(posinal, indice, 1);
        TEdit(Sender).Text := '+'+posinal;
end;
```

indice e posinal são variáveis locais ao procedimento.

Evidentemente, se nossa aplicação utilizar muitos componentes caixas de texto com essas características, ou se muitas aplicações necessitarem dessa funcionalidade, será mais adequado criar um novo componente capaz de incorporar essas funcionalidades, ou criar um template para este componente.

Nos próximos tópicos serão apresentados os procedimentos necessários à criação de templates e à criação desse componente, que neste será representado por uma classe derivada da classe TEdit, e que será denominada TNumEdit.

Criando o Esqueleto do Novo Componente

Para criar o novo componente, você deve executar os seguintes procedimentos:

1. Selecione o item New Component do menu Component, para exibir a caixa de diálogo New Component, mostrada na Figura 21.1.

Nessa caixa de diálogo deverão ser especificados:

- O nome da classe-base (Ancestor Type), da qual o novo componente será derivado por herança. Nesse caso, estaremos criando um novo componente cujas definições serão armazenadas em uma classe derivada de TEdit. Logo, a classe TEdit deverá ser especificada nesse campo. Repare que existem duas opções, correspondentes à VCL (StdCtrls) e CLX (QStdCtrls), e neste exemplo será adotada a primeira opção.
- O nome da classe que representará o novo componente (Class Name). Para essa classe, será atribuído o nome TNumEdit.
- A página da paleta de componentes na qual o componente será inserido (Palette Page). Neste exemplo, criaremos uma paleta denominada Axcel.
- O nome da unit onde será definida a classe do componente (Unit file name). Usaremos o Nome NumEdit.Pas, que será o nome default.
- O nome dos diretórios de pesquisa (Search Path).

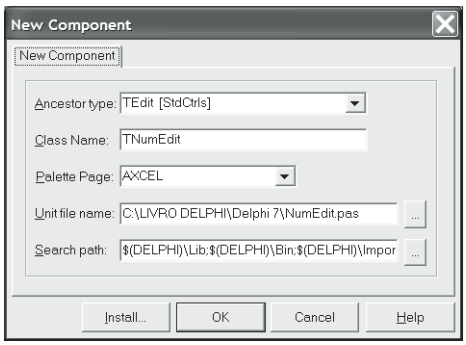


Figura 21.1: A caixa de diálogo New Component.

2. Selecione o botão Ok para fechar essa caixa de diálogo, criar a unit do componente e exibi-la no editor de códigos, como mostrado a seguir.

```
unit NumEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TNumEdit = class(TEdit)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;

procedure Register;

implementation
```

```
procedure Register;
begin
  RegisterComponents('AXCEL', [TNumEdit]);
end;

end.
```

Outra opção seria selecionar o item New do menu File para exibir a caixa de diálogo New Items, o item Component da página New dessa caixa de diálogo e o botão Ok.

Nesse arquivo de código, verifica-se que a nova classe, TNumEdit, será derivada por herança da classe TEdit, como mostra o trecho de código a seguir, em que a mesma é definida:

```
type
  TNumEdit = class(TEdit)
```

Repare ainda que foi definido o procedimento Register, e nesse procedimento é feita uma chamada a um outro procedimento, denominado RegisterComponents.

A procedure RegisterComponents recebe como parâmetro o nome da página da paleta de componentes na qual os novos componentes serão inseridos (se a página indicada não existir, a mesma será criada), e um array de componentes.

Você pode registrar vários componentes de uma única vez para uma única página da paleta de componentes. Nesse caso, o nome de todos os componentes a serem exibidos nessa paleta deverá ser incluído nesse array de componentes.

Caso a unit defina mais de um componente, e esses componentes sejam instalados em paletas distintas, o procedimento Register deverá fazer várias chamadas à procedure RegisterComponents.

Embora o procedimento Register possa conter várias chamadas ao procedimento RegisterComponents, só pode haver um procedimento Register em cada unit.

DEFINIÇÃO DE NOVAS PROPRIEDADES

No capítulo referente à programação orientada a objetos, vimos que uma classe engloba, em uma única entidade, dados (denominados campos) e funções (denominadas métodos). O mesmo se aplica aos componentes (que são um tipo especial de classe), mas nesse caso alguns campos podem ter os seus valores alterados diretamente no Object Inspector, através das suas propriedades.

Podemos concluir, portanto, que uma propriedade serve como um meio de comunicação entre um campo de um componente e o programador, na fase de projeto do aplicativo.

Se considerarmos, por exemplo, um componente Label (da classe TLabel), sabemos que esse componente tem uma propriedade chamada AutoSize, que pode apresentar o valor True ou False. Na realidade, a propriedade AutoSize é um meio de comunicação entre você (o programador) e o campo FAutoSize da

classe TLabel. Você define o valor na propriedade AutoSize, mas na verdade quem armazena esse valor é o campo FAutoSize da classe TLabel.

Se você observar o código-fonte desses componentes no arquivo StdCtrl.pas, verá que o campo FAutoSize está definido na seção private da classe TCustomLabel, e a propriedade AutoSize está definida na seção protected dessa mesma classe. Ocorre, no entanto, que a classe TLabel é derivada, por herança, da classe TCustomLabel (e, conseqüentemente, herda a sua propriedade AutoSize, através da qual acessará o campo FAutoSize). Além disso, a classe TLabel redefine a propriedade AutoSize na seção published da classe que representa o componente, seção esta cuja definição será vista no tópico a seguir.

Se você observar o diagrama de hierarquia de classes da VCL, verá que existem diversas classes cujo nome começa com a palavra "TCustom", e que declaram algumas propriedades como protected, tornadas published em classes delas derivadas por herança.

Criando uma Nova Propriedade

Neste tópico serão apresentados os procedimentos necessários à criação de uma nova propriedade. Conforme já foi dito anteriormente, o componente NumEdit poderá ser usado, opcionalmente, para permitir apenas a digitação de números inteiros, e isso será definido mediante a atribuição de um valor adequado a uma propriedade do componente que chamaremos de Numerico. Essa propriedade será do tipo booleana e poderá, portanto, receber apenas os valores True e False.

Normalmente, conforme dito no tópico anterior, o valor de uma propriedade é armazenado em um campo privado da classe e, dessa maneira, a propriedade serve apenas como uma interface ou meio de comunicação entre o campo e o desenvolvedor.

Convencionalmente, os campos internos dos componentes costumam ter o mesmo nome das propriedades, precedidos pela letra F (de Field). Nesse caso, portanto, se a propriedade será denominada Numerico, o campo respectivo poderá ser denominado FNumerico (repare que não estamos acentuando esses nomes, seguindo portanto as mesmas regras estabelecidas para a definição de nomes de variáveis).

A criação da propriedade propriamente dita, no entanto, será feita na seção published da classe do componente, mediante a inclusão de uma linha de código que apresenta a seguinte sintaxe:

```
property nome_propriedade : tipo read metodo_ leitura write metodo_ escrita;
```

Como pode ser visto na definição anterior, uma propriedade pode ter um método de leitura e um método de escrita. Quando se omite o nome do método de escrita, a propriedade é do tipo read-only (apenas de leitura) e seu valor pode ser apenas lido.

Existem situações, no entanto, que, em vez de se definir um método de leitura e um método de escrita, coloca-se em seu lugar o nome do campo em que o valor da propriedade será armazenado. Nesses casos, diz-se que a propriedade acessa diretamente o valor de um campo, sem que se use qualquer método específico.

No caso da propriedade Numerico, sua definição será inicialmente feita mediante a inclusão da seguinte linha de código na seção published da classe:

```
property Numerico : boolean read FNumerico write FNumerico;
```

A definição da classe, portanto, passaria a ter a forma apresentada a seguir.

```
TNumEdit = class(TEdit)
  private
    { Private declarations }
    FNumerico : boolean;
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
    property Numerico : boolean read FNumerico write FNumerico;
  end;
```

Não se esqueça de incluir, na seção private da classe que representa o componente, a definição do campo que armazenará internamente o valor da propriedade.

Instalando o Novo Componente

Antes de incrementar ainda mais o nosso componente, vamos instalá-lo e analisar o funcionamento da propriedade recém-criada.

Para instalar o componente, selecione o item Install Component do menu Component. Será exibida a caixa de diálogo Install Component, apresentada na figura a seguir. Nesse caso, o componente será instalado no pacote dclusr.dpk (para instalar o componente em um novo pacote, selecione a guia Into new package e preencha corretamente seus diversos campos).

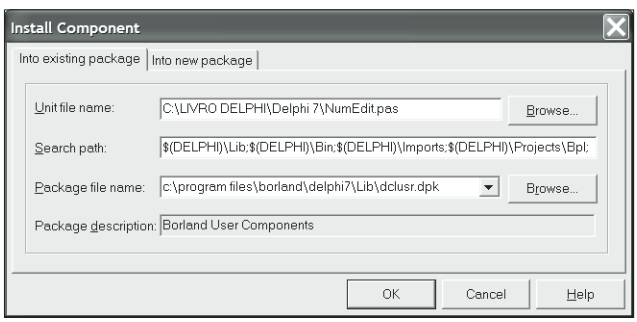


Figura 21.2: A caixa de diálogo Install Component.

Após preencher corretamente as informações dessa caixa de diálogo, selecione o botão OK. Será exibida a caixa de diálogo de confirmação mostrada na figura a seguir.

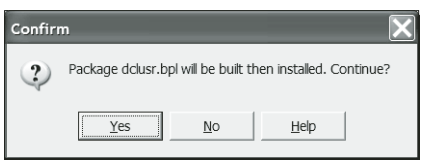


Figura 21.3: A caixa de diálogo de confirmação.

O componente será instalado no pacote especificado, como mostra a figura a seguir.

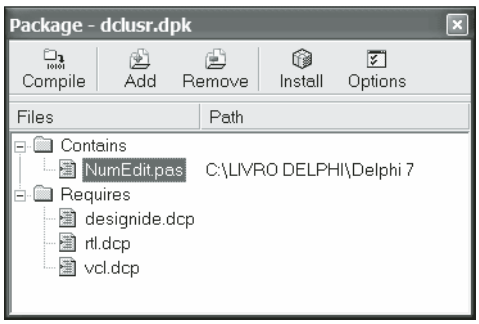


Figura 21.4: Instalando o componente em um pacote.

Durante a instalação serão exibidas algumas mensagens informativas. Basta selecionar o botão Ok dessas caixas de diálogo para dar prosseguimento ao processo.

Nesse caso, o componente foi inserido no pacote dclusr.dpk, que possui exatamente esta finalidade – armazenar a definição de componentes criados pelo desenvolvedor.

O componente estará instalado e estará disponível na página Axcel da paleta de componentes, como mostra a figura a seguir.

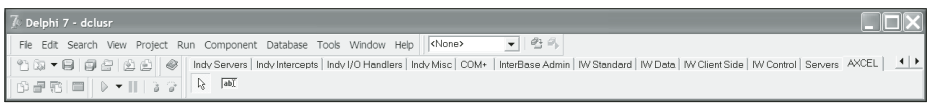


Figura 21.5: A nova página da paleta de componentes.

Caso o componente não tenha sido instalado, selecione os botões Compile e Install na janela Package do pacote correspondente.

Repare que, como não foi definido um bitmap específico para o componente, o bitmap do seu componente ancestral foi automaticamente utilizado (nesse caso, o bitmap definido para o componente TEdit). Posteriormente serão apresentados os procedimentos necessários à definição de um bitmap para o novo componente.

Se você colocar o cursor do mouse sobre o bitmap que identifica o componente na paleta, será exibida uma string de auxílio com o nome do componente (no caso, NumEdit).

Coloque o componente em um formulário e verifique que a nova propriedade já está disponível no Object Inspector, como mostra a figura a seguir.

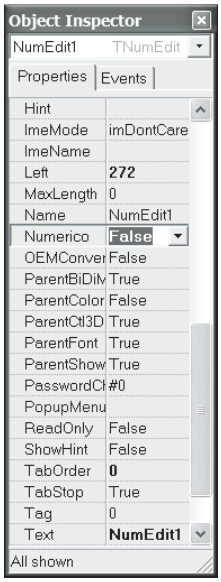


Figura 21.6: Visualizando a nova propriedade no Object Inspector.

Repare que o valor default dessa propriedade é False. Para alterar esse valor default, você deve sobrecarregar o construtor da classe-base e definir o novo valor.

Isso será feito da maneira descrita no tópico apresentado a seguir.

Evidentemente, a alteração do valor dessa propriedade ainda não provoca nenhum efeito, pois o componente ainda não foi codificado para reagir a alteração de valores da propriedade Numerico. Nos próximos tópicos, esses aspectos serão tratados em maiores detalhes.

Sobrecarregando o Método Construtor da Classe Ancestral do Componente

A definição de um novo construtor, sobrecarregando o construtor da sua classe ancestral e utilizando o conceito de polimorfismo (apresentado no capítulo referente à programação orientada a objetos), é feita mediante a utilização do seguinte trecho de código:

```
type
  TNumEdit = class(TEdit)
  private
    { Private declarations }
    FNumerico : boolean;
  protected
    { Protected declarations }
  public
    { Public declarations }
    constructor Create(AOwner: TComponent);override;
  published
    { Published declarations }
    property Numerico : boolean read FNumerico write FNumerico;
  end;
```

A implementação do construtor deve ser feita na seção implementation da unit, como apresentado a seguir:

```
constructor TNumEdit.Create(AOwner: TComponent);
begin
    inherited Create(AOwner);
    FNumerico := True;
    Text := '0';
end;
```

Repare que a primeira coisa que o construtor faz é chamar o método construtor da classe-base. Em seguida, define as propriedades Numerico e Text como True e 0, respectivamente (afinal de contas, não seria lógico que o valor default da propriedade Text desse componente fosse "NumEdit1").

Reinstale novamente o componente. Você verá que agora o valor default da propriedade Numerico será igual a True.

Para reinstalar o componente, você deve executar os seguintes procedimentos:

1. Selecionar o item Install Packages do menu Components. Será exibida a caixa de diálogo Project Options, na qual deverá ser selecionado o item Borland Delphi User, conforme indicado na figura a seguir.

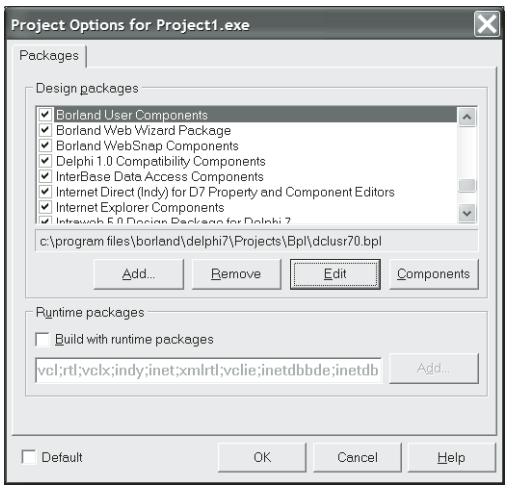


Figura 21.7: A caixa de diálogo Project Options.

2. Selecione o botão Edit dessa caixa de diálogo. Será exibida uma caixa de diálogo solicitando uma confirmação, e indicando que a caixa de diálogo Project Options será fechada.
3. Será exibida a caixa de diálogo do editor de pacotes. Selecione o botão Compile dessa caixa de diálogo, para recompilar o pacote.

Dessa maneira, a partir desse momento, a propriedade Numerico terá o valor default igual a True e a proprieade Text o valor '0'.

Entretanto, se você observar a descrição textual de um formulário após a inclusão desse componente, verá que a descrição do objeto é feita da seguinte maneira (os valores numéricos podem variar um pouco):

```
object NumEdit1: TNumEdit
    Left = 208
    Top = 168
```

```
    Width = 121
    Height = 21
    TabOrder = 0
    Text = '0'
    Numerico = True
  End
```

Para que o valor default da propriedade Numerico do componente TNumEdit não seja armazenado na descrição textual do formulário que o contém, torna-se necessário redefinir a declaração da propriedade na definição da classe, como mostra o trecho de código a seguir.

```
property Numerico : boolean read FNumerico write Fnumerico default True;
```

Repare que definir um valor default para a propriedade (ou campo) é diferente de definir um valor default para a sua descrição textual no formulário.

A declaração de um valor default para o campo é feita no construtor, ao passo que a definição de um valor default para definir a inclusão ou não do valor da propriedade na descrição textual do formulário que contém o componente é feita na própria definição da propriedade.

Dessa maneira, se o valor da propriedade Numerico for igual a True, sua definição não será mais incluída na descrição textual do formulário.

Redefinindo Métodos da Classe-base

Nesse componente, como desejamos que o usuário só possa digitar valores numéricos inteiros caso a propriedade Numerico seja igual a True, precisamos redefinir os método KeyPress e Change existentes na classe-base, a serem implementados como mostrados nos trechos de código apresentados a seguir.

```
procedure TNumEdit.KeyPress(var Key : Char);
begin
     if FNumerico then
     begin
        if (not (key in ['0'..'9','-','+',#8]))or((key in ['+','-']) and (Pos(key,Text)>0))
        then key := #0;
     end;
     inherited KeyPress(Key);
end;

procedure TNumEdit.Change;
var
   indice : word;
   posinal : string;
   valorconvertido : integer;
begin
    if FNumerico then
    begin
       indice := Pos('-',Text);
       if indice > 1
       then
       begin
          posinal := Text;
          System.Delete(posinal, indice, 1);
          Text := '-'+posinal;
       end;
       indice := Pos('+',Text);
       if indice > 1
```

```
        then
        begin
           posinal := Text;
           System.Delete(posinal, indice, 1);
           Text := '+'+posinal;
        end;
        try
          valorconvertido := StrToInt(Text);
        except
            on EConvertError do
            begin
                Numerico := False;
            end;
        end;
    end;
    inherited Change;
end;
```

Com a implementação desse método definida dessa maneira, caso o usuário insista em digitar um caractere inválido no componente na fase de projeto do aplicativo, o valor da propriedade Numerico será redefinido como false.

A redefinição da classe por completa é apresentada a seguir.

```
type
  TNumEdit = class(TEdit)
  private
    { Private declarations }
    FNumerico : boolean;
  protected
    { Protected declarations }
    procedure KeyPress(var Key : Char);override;
    procedure Change;override;
  public
    { Public declarations }
    constructor Create(AOwner: TComponent);override;  published
    { Published declarations }
    property Numerico : boolean read FNumerico write FNumerico default True;
end;
```

A partir de então, se o valor da propriedade Numerico for igual a True, apenas valores numéricos poderão ser digitados no componente durante a execução de um aplicativo que o utilize.

Dessa maneira, a utilização do componente TNumEdit evita que se tenha de codificar os procedimentos associados aos eventos OnKeyPress e OnChange para cada caixa de texto, e é exatamente este o objetivo de se criar novos componentes – automatizar tarefas realizadas com freqüência no desenvolvimento dos seus aplicativos.

Além disso, da forma como foi definido, esse componente ainda pode ser utilizado como uma caixa de texto comum, bastando para isso que se defina como False o valor da sua propriedade Numerico.

Considere agora a situação em que se prefere dar ao usuário a capacidade de definir o que deve ser feito caso a conversão descrita anteriormente não seja possível. O que fazer nesse caso? Que tal criarmos um evento que deve ser disparado quando isso ocorrer, e deixar a cargo do usuário a tarefa de tomar uma decisão? Embora este exemplo possa parecer meio simplório, ao menos ajuda a esclarecer didaticamente o processo de definição de novos eventos para um componente. Além disso, veremos como usar métodos de leitura e escrita na definição de novas propriedades.

Definindo um Novo Evento Para o Componente

Neste tópico definiremos um novo evento para o componente TNumEdit, cujo procedimento associado será executado sempre que o usuário alterar para True o valor da propriedade Numerico do componente, mas o texto exibido pelo componente não puder ser convertido em número.

Inicialmente, deve-se considerar o fato de que um evento é, na realidade, uma propriedade. Isso mesmo: um evento é uma propriedade especial, que aponta para uma função a ser executada!

O tipo mais básico de evento é o do tipo TNotifyEvent, cujo procedimento associado terá como parâmetro uma única variável chamada Sender, da classe TObject. No nosso caso, definiremos um evento chamado OnErroConversao, alterando a definição da nossa classe como a seguir.

```
type
TNumEdit = class(TEdit)
  private
    { Private declarations }
    FNumerico : boolean;
    FOnErroConversao : TNotifyEvent;
  protected
    { Protected declarations }
    procedure KeyPress(var Key : Char);override;
    procedure Change;override;
  public
    { Public declarations }
    constructor Create(AOwner: TComponent);override;
  published
    { Published declarations }
    property Numerico : boolean read FNumerico write Fnumerico default True;
    property OnErroConversao : TNotifyEvent read FOnErroConversao write FOnErroConversao;
end;
```

Repare que, por ser uma propriedade, o evento OnErroConversão também possui um campo interno associado, denominado FOnErroConversao.

No entanto, esse evento deverá ser disparado quando o usuário alterar o valor da propriedade Numerico, e apenas se isso ocorrer durante a execução do aplicativo. Dessa forma, devemos definir um método de escrita para essa variável, de modo a tratar esse fato.

Um método de escrita deve ter um único parâmetro, e este deve ser do mesmo tipo da propriedade acessada. Podemos então definir, através do seguinte código, um método denominado AtribuiValor para a classe TNumEdit:

```
procedure TNumEdit.AtribuiValor(NovoValor : boolean);
var
   valorconvertido : Integer;
begin
    FNumerico := NovoValor;
    if not(NovoValor) then exit;
    try
        valorconvertido := StrToInt(Text);
    except
        on EConvertError do
        begin
           if (csdesigning in ComponentState) then
               Text := '0'
           else
```

```
                if Assigned(FOnErroConversao) then FOnErroConversao(Self);
            end;
        end;
end;
```

Nesse procedimento, verifica-se se o valor da propriedade Numerico está sendo alterada na fase de projeto do aplicativo, verificando-se se a propriedade ComponentState (que é um conjunto) contém entre seus elementos a constante csdesigning – que indica que a propriedade está sendo alterada na fase de projeto do aplicativo.

A classe terá, conseqüentemente, a sua definição alterada para:

```
type
TNumEdit = class(TEdit)
  private
    { Private declarations }
    FNumerico : boolean;
    FOnErroConversao : TNotifyEvent;
  protected
    { Protected declarations }
    procedure KeyPress(var Key : Char);override;
    procedure Change;override;
    procedure AtribuiValor(NovoValor : boolean);
  public
    { Public declarations }
    constructor Create(AOwner: TComponent);override;
  published
    { Published declarations }
    property Numerico : boolean read FNumerico write AtribuiValor default True;
    property OnErroConversao : TNotifyEvent read FOnErroConversao write FOnErroConversao;
end;
```

Repare que agora o valor da propriedade Numerico é escrito no campo FNumerico através de um método chamado AtribuiValor.

Você pode agora recompilar o pacote que contém este novo componente, incluir um componente NumEdit no formulário e definir, da seguinte maneira, o procedimento associado ao seu evento OnErroConversao:

```
procedure TForm1.NumEdit1ErroConversao(Sender: TObject);
begin
     NumEdit1.text := '0';
end;
```

Dessa maneira, sempre que neste exemplo ocorrer um erro de conversão durante a execução do aplicativo, o valor da propriedade Text de NumEdit1 será igual a '0'. Isso foi definido pelo usuário do componente, que poderá não ter acesso ao seu código-fonte, mas poderá decidir o que deve ser feito pelo programa quando houver um erro de conversão.

O Tipo TNotifyEvent

O tipo TNotifyEvent está definido da seguinte maneira na unit Classes:

```
type TNotifyEvent = procedure (Sender: TObject) of object;
```

Caso você queira, pode definir outros tipos de eventos, como descrito a seguir.

```
type TMeuEvento = procedure of object;
```

Nesse caso, se essa definição fosse incluída antes da definição da classe TNumEdit, você poderia definir FOnErroConversao como:

```
FOnErroConversao : TMeuEvento;
```

E a propriedade OnErroConversao teria de ser redefinida como:

```
property OnErroConversao : TMeuEvent read FOnErroConversao write FOnErroConversao;
```

No método AtribuiEvento, a chamada do procedimento associado ao evento seria feita da seguinte maneira:

```
if Assigned(FOnErroConversao) then FOnErroConversao;
```

Repare que, nesse caso, não existirão mais parâmetros a passar, pois você usou o tipo TMeuEvento em vez de TNotifyEvent! Nada o impede, no entanto, de definir outros eventos com parâmetros distintos. Repare os procedimentos associados a diversos eventos de um formulário, e verifique as diferenças existentes em suas listas de parâmetros.

Definindo um Método de Leitura Para uma Propriedade

Nos tópicos anteriores foram apresentados os procedimentos necessários à definição de um método de escrita para uma propriedade. A definição de um método de leitura segue procedimentos semelhantes, devendo-se considerar, no entanto, que:

- Enquanto um método de escrita é um procedimento, um método de leitura deve ser uma função, cujo tipo de retorno deve ser o mesmo tipo definido para a propriedade.
- Enquanto um método de escrita deve possuir um parâmetro, uma função não deve possuir nenhum (excetuando-se o caso de propriedades definidas como arrays).

Dessa maneira, um método de leitura definido de maneira bastante simples, com o objetivo de obter o valor da propriedade Numerico, poderia ser implementado da seguinte maneira:

```
function TNumEdit.LeValor : boolean;
begin
    result := FNumerico;
end;
```

Repare que esse método é bastante simples, e pouco acrescenta ao método de acesso direto, pois apenas retorna o valor armazenado no campo FNumerico da classe. Nosso objetivo, contudo, foi mostrar os procedimentos necessários à definição de um método de leitura para uma propriedade de um componente.

No Delphi, os métodos de leitura e de escrita costumam ter os seus nomes iniciados com as palavras "Get" e "Set", respectivamente. Esta é, no entanto, apenas uma convenção, pois, conforme mostrado anteriormente, os métodos podem ter qualquer nome válido, de acordo com as regras da linguagem Object Pascal. De qualquer forma, nos próximos tópicos seguiremos as convenções do Delphi.

Apresenta-se a seguir o código final da unit que define a classe TNumEdit:

```
unit NumEdit;

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
TNumEdit = class(TEdit)
  private
    { Private declarations }
    FNumerico : boolean;
    FOnErroConversao : TNotifyEvent;
  protected
    { Protected declarations }
    procedure KeyPress(var Key : Char);override;
    procedure Change;override;
    procedure AtribuiValor(NovoValor : boolean);
    function LeValor : boolean;
  public
    { Public declarations }
    constructor Create(AOwner: TComponent);override;
  published
    { Published declarations }
    property Numerico : boolean read Levalor write AtribuiValor default True;
    property OnErroConversao : TNotifyEvent read FOnErroConversao write FOnErroConversao;
end;

procedure Register;

implementation

function TNumEdit.LeValor : boolean;
begin
    result := FNumerico;
end;

procedure TNumEdit.AtribuiValor(NovoValor : boolean);
var
   valorconvertido : Integer;
begin
    FNumerico := NovoValor;
    if not(NovoValor) then exit;
    try
        valorconvertido := StrToInt(Text);
    except
        on EConvertError do
        begin
           if (csdesigning in ComponentState) then
               Text := '0'
           else
               if Assigned(FOnErroConversao) then FOnErroConversao(Self);
         end;
      end;
end;

procedure TNumEdit.KeyPress(var Key : Char);
begin
     if FNumerico then
     begin
        if (not (key in ['0'..'9','-','+',#8]))or((key in ['+','-']) and (Pos(key,Text)>0))
        then key := #0;
     end;
```

```
     inherited KeyPress(Key);
end;

procedure TNumEdit.Change;
var
   indice : word;
   posinal : string;
   valorconvertido : integer;
begin
    if FNumerico then
    begin
       indice := Pos('-',Text);
       if indice > 1
       then
       begin
          posinal := Text;
          System.Delete(posinal, indice, 1);
          Text := '-'+posinal;
       end;
       indice := Pos('+',Text);
       if indice > 1
       then
       begin
          posinal := Text;
          System.Delete(posinal, indice, 1);
          Text := '+'+posinal;
       end;
       try
         valorconvertido := StrToInt(Text);
       except
           on EConvertError do
           begin
               Numerico := False;
           end;
       end;
    end;
    inherited Change;
end;

constructor TNumEdit.Create(AOwner: TComponent);
begin
    inherited Create(AOwner);
    FNumerico := True;
    Text := '0';
end;

procedure Register;
begin
  RegisterComponents('KnowHow', [TNumEdit]);
end;

end.
```

KNOW-HOW EM: CRIAÇÃO DE COMPONENTES ASSOCIADOS A BANCOS DE DADOS

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 7.

- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi 7.
- Fundamentos básicos da criação de componentes em Delphi 7.

METODOLOGIA

- Apresentação do problema: Associação de um componente ao valor armazenado em um campo de uma tabela de um banco de dados.

TÉCNICA

- Apresentação dos procedimentos necessários à criação de componentes associados a bancos de dados.

Apresentação do Problema

Nos tópicos anteriores foram apresentados os procedimentos necessários à criação de novos componentes com o Delphi 7.

Existem situações, no entanto, em que o valor exibido pelo componente deve refletir o valor armazenado em um campo de uma tabela. Além disso, qualquer alteração feita no valor exibido pelo componente deve ser refletida no valor armazenado na tabela.

Nos tópicos subseqüentes, serão apresentados os procedimentos necessários à conversão do componente TNumEdit, descrito e desenvolvido nos tópicos anteriores, em um componente associado a um banco de dados (ou, como costuma-se denominar na linguagem object Pascal, um componente "data-aware") que será denominado TDBNumEdit.

Criando o Esqueleto do Novo Componente

O novo componente será denominado TDBNumEdit, e será derivado diretamente do componente TNumEdit.

Para criar o esqueleto desse novo componente, basta executar os procedimentos descritos no início deste capítulo, e salvar a unit na qual será armazenada a definição do componente com o nome DBNumEdit.pas.

Inicialmente, esse arquivo apresentará o código reproduzido a seguir.

```
unit DBNumEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, NumEdit;

type
  TDBNumEdit = class(TNumEdit)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
```

```
    end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('AXCEL', [TDBNumEdit]);
end;

end.
```

Como já deve ser do seu conhecimento, os componentes associados a um banco de dados costumam apresentar as seguintes propriedades:

- DataSource: Que identifica o objeto da classe TDataSource ao qual o componente está vinculado.
- DataField: O nome do campo da tabela cujo valor do registro corrente será exibido e/ou editado pelo componente.

A princípio, seria natural que se criassem dois campos internos, denominados FDataSource (do tipo TDataSource) e FDataField (do tipo string) para armazenarem internamente os valores dessas propriedades.

Ocorre no entanto que, para simplificar o desenvolvimento de componentes associados a bancos de dados, a Borland incorporou à biblioteca de classes do Delphi uma classe chamada TFieldDataLink, a partir da qual poderemos criar um objeto responsável pela associação do componente ao banco de dados.

A classe TFieldDataLink possui, entre outras, as seguintes propriedades:

- DataSource: Que identificará o componente da classe TDataSource ao qual o nosso componente será vinculado.
- FieldName: Que identificará o nome do campo ao qual o nosso componente será vinculado.

Conseqüentemente, em vez de criarmos campos internos para armazenar os valores das propriedades DataSource e DataField do nosso componente, podemos utilizar as propriedades DataSource e FieldName de um objeto da classe TFieldDataLink (a ser criado e destruído pelo nosso componente) nos métodos de acesso dessas propriedades.

Além disso, como a classe TFieldDataLink está definida na unit dbctrls, o nome dessa unit deverá ser incluído na cláusula uses da unit que define a classe TBDNumEdit, e o mesmo se aplica à unit db, na qual está definida a classe TDataSource.

Definindo as Novas Propriedades Para o Componente

Conforme descrito anteriormente, as propriedades DataSource e DataField do componente serão armazenadas em um objeto da classe TFieldDataLink.

Conseqüentemente, precisamos definir um campo interno como um objeto dessa classe, incluindo-se a seguinte linha de código na seção private da classe que define o componente:

```
FDataLink : TFieldDataLink;
```

Além disso, como as propriedades DataSource e DataField serão na realidade armazenadas em propriedades do objeto FDataLink, precisamos definir métodos de leitura e de escrita para essas propriedades, o que pode ser feito incluindo-se as seguintes linhas de código na seção private da classe que define o componente (e usando as denominações Get para leitura e Set para escrita):

```
function GetDataSource : TDataSource;
Procedure SetDataSource(Value : TDataSource);
function GetDataField : string;
procedure SetDataField(Value : string);
```

As propriedades são definidas incluindo-se as seguintes linhas de código na seção public da classe que define o componente:

```
property DataSource : TDataSource read GetDataSource write SetDataSource;
property DataField : string read GetDataField write SetDataField;
```

Os métodos de leitura e escrita deverão ser implementados como mostrados nos trechos de código apresentados a seguir.

```
function TDBNumEdit.GetDataSource : TDataSource;
begin
   Result := FDataLink.DataSource;
end;

procedure TDBNumEdit.SetDataSource(Value : TDataSource);
begin
   FDataLink.DataSource := Value;
end;

function TDBNumEdit.GetDataField : string;
begin
   Result := FDataLink.FieldName;
end;

procedure TDBNumEdit.SetDataField(Value : string);
begin
   FDataLink.FieldName := Value;
end;
```

Conforme descrito anteriormente, nosso componente será responsável por criar e destruir o objeto FDataLink, o que deve ser feito sobrecarregando-se os métodos construtor e destrutor do componente.

No método construtor deverão ser incluídas as seguintes linhas de código:

```
inherited Create(AOwner);
FDataLink := TFieldDataLink.Create;
```

Lembre-se de que o método construtor é um método virtual, e que estamos sobrecarregando o método construtor da classe-base.

No método destrutor deverão ser incluídas as seguintes linhas de código:

```
FDataLink.Free;
inherited Destroy;
```

Apresenta-se a seguir o código da unit que define a classe TDBNumEdit, após terem sido feitas as alterações descritas anteriormente.

```
unit DBNumEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, NumEdit, dbCtrls, db;

type
  TDBNumEdit = class(TNumEdit)
  private
    { Private declarations }
    FDataLink : TFieldDataLink;
    function GetDataSource : TDataSource;
    procedure SetDataSource(Value : TDataSource);
    function GetDataField : string;
    procedure SetDataField(Value : string);
  protected
    { Protected declarations }
  public
    { Public declarations }
    constructor Create(AOwner: TComponent);override;
    destructor Destroy;override;
  published
    { Published declarations }
    property DataSource : TDataSource read GetDataSource write SetDataSource;
    property DataField : string read GetDataField write SetDataField;
  end;

procedure Register;

implementation

constructor TDBNumEdit.Create(AOwner: TComponent);
begin
    FDataLink := TFieldDataLink.Create;
    inherited Create(AOwner);
end;

destructor TDBNumEdit.Destroy;
begin
    FDataLink.Free;
    inherited Destroy;
end;

function TDBNumEdit.GetDataSource : TDataSource;
begin
   Result := FDataLink.DataSource;
end;

procedure TDBNumEdit.SetDataSource(Value : TDataSource);
begin
   FDataLink.DataSource := Value;
end;

function TDBNumEdit.GetDataField : string;
begin
   Result := FDataLink.FieldName;
end;

procedure TDBNumEdit.SetDataField(Value : string);
begin
   FDataLink.FieldName := Value;
end;
```

```
procedure Register;
begin
  RegisterComponents('KnowHow', [TDBNumEdit]);
end;

end.
```

Refletindo Alterações Feitas no Campo

Uma das características do nosso componente deve ser a capacidade de refletir as alterações feitas no campo da tabela a ela associada.

Para facilitar a implementação dessa característica, a classe TFieldDataLink tem o evento OnDataChange, cujo procedimento associado é executado sempre que o valor armazenado no campo é alterado.

Entretanto, como essa classe não define um componente, seus eventos não podem ser acessados através do Object Inspector, e a associação de um procedimento a esse evento deve ser feita mediante uma adequada codificação.

Nesse caso, devem-se executar os seguintes procedimentos:

1. Definir um método para o componente, a ser associado ao evento OnDataChange do objeto FDataLink, da classe TFieldDataLink.

Neste exemplo, criaremos um método chamado DataChange, responsável por atualizar o texto exibido pelo componente. Esse método deve ser declarado na seção private da classe que define o componente, e implementado da seguinte maneira:

```
procedure TDBNumEdit.DataChange(Sender : TObject);
begin
      try
          Text := FDataLink.Field.Text;
      except
          Text := '0';
      end;
end;
```

Incluímos um tratamento de exceção para o caso de não haver sucesso na associação desejada.

2. Associar esse procedimento ao evento OnDataChange do objeto FDataLink, o que pode ser feito incluindo-se a seguinte linha de código no método construtor do componente:

```
FDataLink.OnDataChange := DataChange;
```

Apresenta-se a seguir o código da unit que define a classe TDBNumEdit, após terem sido feitas as alterações descritas anteriormente.

```
unit DBNumEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, NumEdit, dbCtrls, db;
```

```
type
  TDBNumEdit = class(TNumEdit)
  private
    { Private declarations }
    FDataLink : TFieldDataLink;
    function GetDataSource : TDataSource;
    procedure SetDataSource(Value : TDataSource);
    function GetDataField : string;
    procedure SetDataField(Value : string);
    procedure DataChange(Sender : TObject);
  protected
    { Protected declarations }
  public
    { Public declarations }
    constructor Create(AOwner: TComponent);override;
    destructor Destroy;override;
  published
    { Published declarations }
    property DataSource : TDataSource read GetDataSource write SetDataSource;
    property DataField : string read GetDataField write SetDataField;
  end;

procedure Register;

implementation

procedure TDBNumEdit.DataChange(Sender : TObject);
begin
      try
          Text := FDataLink.Field.Text;
      except
          Text := '0';
      end;
end;

constructor TDBNumEdit.Create(AOwner: TComponent);
begin
    FDataLink := TFieldDataLink.Create;
    inherited Create(AOwner);
    FDataLink.OnDataChange := DataChange;
end;

destructor TDBNumEdit.Destroy;
begin
    FDataLink.Free;
    inherited Destroy;
end;

function TDBNumEdit.GetDataSource : TDataSource;
begin
   Result := FDataLink.DataSource;
end;

procedure TDBNumEdit.SetDataSource(Value : TDataSource);
begin
   FDataLink.DataSource := Value;
end;

function TDBNumEdit.GetDataField : string;
begin
   Result := FDataLink.FieldName;
end;

procedure TDBNumEdit.SetDataField(Value : string);
```

```
begin
   FDataLink.FieldName := Value;
end;

procedure Register;
begin
  RegisterComponents('KnowHow', [TDBNumEdit]);
end;

end.
```

Refletindo Alterações Feitas no Componente

Uma das características do nosso novo componente deve ser a capacidade de refletir no campo associado as alterações feitas no valor por ele exibido.

Para facilitar a implementação dessa característica, a classe TFieldDataLink tem o evento OnUpdateData, cujo procedimento associado é executado sempre que o valor armazenado no campo deve ser modificado.

Nesse caso a associação de um procedimento a esse evento também deve ser feita mediante uma adequada codificação.

Nesse caso, devem-se executar os seguintes procedimentos:

1. Definir um método para o componente, a ser associado ao evento OnUpdateData do objeto FDataLink, da classe TFieldDataLink.

Neste exemplo, criaremos um método chamado UpdateData, responsável por atualizar no campo o texto exibido pelo componente. Esse método deve ser declarado na seção private da classe que define o componente, e implementado da seguinte maneira:

```
procedure TDBNumEdit.UpdateData(Sender : TObject);
begin
       try
          FDataLink.Field.Text := Text;
       except
          begin
             FDataLink.Edit;
             FDataLink.Field.Text := Text;
          end;
       end;
end;
```

Nesse método, caso ocorra uma exceção na tentativa de se atualizar o campo, o objeto FDataLink é colocado em modo de edição e a alteração é efetivada.

2. Associar esse procedimento ao evento OnUpdateData do objeto FDataLink, o que pode ser feito incluindo-se a seguinte linha de código no método construtor do componente:

```
FDataLink.OnUpdateData := UpdateData;
```

3. Sobrecarregar o método Change da classe-base, de maneira que o valor armazenado no campo seja alterado sempre que o usuário alterar o texto exibido pelo componente.

A implementação do método Change deve ser feita da maneira apresentada a seguir.

Repare que esse método é responsável por chamar os métodos Modified e UpdateRecord do objeto FDataLink, este responsável por gerar o seu evento OnUpdateData e conseqüentemente executar o procedimento a ele associado.

```
procedure TDBNumEdit.Change;
begin
    FDataLink.ModiFied;
    inherited Change;
    try
       FDataLink.Edit;
       FDataLink.UpdateRecord;
    except
    end;
end;
```

Apresenta-se a seguir o código da unit que define a classe TDBNumEdit, após terem sido feitas as alterações descritas anteriormente.

```
unit DBNumEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, NumEdit, dbCtrls, db;

type
  TDBNumEdit = class(TNumEdit)
  private
    { Private declarations }
    FDataLink : TFieldDataLink;
    function GetDataSource : TDataSource;
    procedure SetDataSource(Value : TDataSource);
    function GetDataField : string;
    procedure SetDataField(Value : string);
    procedure DataChange(Sender : TObject);
    procedure UpdateData(Sender : TObject);
    procedure Change;override;
  protected
    { Protected declarations }
  public
    { Public declarations }
    constructor Create(AOwner: TComponent);override;
    destructor Destroy;override;
  published
    { Published declarations }
    property DataSource : TDataSource read GetDataSource write SetDataSource;
    property DataField : string read GetDataField write SetDataField;
  end;

procedure Register;

implementation

procedure TDBNumEdit.Change;
begin
    FDataLink.ModiFied;
    inherited Change;
    try
       FDataLink.Edit;
       FDataLink.UpdateRecord;
    except
```

```
    end;
end;

procedure TDBNumEdit.UpdateData(Sender : TObject);
begin
       try
          FDataLink.Field.Text := Text;
       except
          begin
             FDataLink.Edit;
             FDataLink.Field.Text := Text;
          end;
       end;
end;

procedure TDBNumEdit.DataChange(Sender : TObject);
begin
      try
          Text := FDataLink.Field.Text;
      except
          Text := '0';
      end;
end;

constructor TDBNumEdit.Create(AOwner: TComponent);
begin
    FDataLink := TFieldDataLink.Create;
    inherited Create(AOwner);
    FDataLink.OnDataChange := DataChange;
    FDataLink.OnUpdateData := UpdateData;
end;

destructor TDBNumEdit.Destroy;
begin
    FDataLink.Free;
    inherited Destroy;
end;

function TDBNumEdit.GetDataSource : TDataSource;
begin
   Result := FDataLink.DataSource;
end;

procedure TDBNumEdit.SetDataSource(Value : TDataSource);
begin
   FDataLink.DataSource := Value;
end;

function TDBNumEdit.GetDataField : string;
begin
   Result := FDataLink.FieldName;
end;

procedure TDBNumEdit.SetDataField(Value : string);
begin
   FDataLink.FieldName := Value;
end;

procedure Register;
begin
  RegisterComponents('KnowHow', [TDBNumEdit]);
end;

end.
```

Notificando o Componente da Remoção de um DataSource

Da forma como nosso componente está implementado, se removermos o componente DataSource do formulário, a propriedade DataSource do componente não apresentará nenhum valor (o que está correto) mas o valor da propriedade DataField não é alterado (o que está errado).

A solução desse problema consiste em sobrecarregar o método Notification do seu componente ancestral.

Esse método é executado sempre que um componente é inserido ou movido do formulário, e recebe como parâmetros:

- Um ponteiro para o componente que foi inserido ou removido.
- Uma variável do tipo TOperation, que identifica a operação realizada sobre o componente.

Neste exemplo, devemos remover o valor exibido na propriedade DataField sempre que o DataSource associado for removido, o que pode ser feito implementando-se da seguinte maneira o método Notification do componente:

```
procedure TDBNumEdit.Notification(AComponent : TComponent;Operation : TOperation);
begin
     if (Operation = opRemove)and(AComponent is TDataSource)and(FDataLink.DataSource = nil)
     then FDataLink.FieldName := '';
end;
```

Apresenta-se a seguir o código da unit que define a classe TDBNumEdit, após terem sido feitas as alterações descritas anteriormente.

```
unit DBNumEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, NumEdit, dbCtrls, db;

type
  TDBNumEdit = class(TNumEdit)
  private
    { Private declarations }
    FDataLink : TFieldDataLink;
    function GetDataSource : TDataSource;
    procedure SetDataSource(Value : TDataSource);
    function GetDataField : string;
    procedure SetDataField(Value : string);
    procedure DataChange(Sender : TObject);
    procedure UpdateData(Sender : TObject);
    procedure Change;override;
  protected
    { Protected declarations }
    procedure Notification(AComponent : TComponent;Operation : TOperation);override;
  public
    { Public declarations }
    constructor Create(AOwner: TComponent);override;
    destructor Destroy;override;
  published
    { Published declarations }
    property DataSource : TDataSource read GetDataSource write SetDataSource;
```

```
    property DataField : string read GetDataField write SetDataField;
  end;

procedure Register;

implementation

procedure TDBNumEdit.Notification(AComponent : TComponent;Operation : TOperation);
begin
     if (Operation = opRemove)and(AComponent is TDataSource)and(FDataLink.DataSource = nil)
     then FDataLink.FieldName := '';
end;

procedure TDBNumEdit.Change;
begin
    FDataLink.ModiFied;
    inherited Change;
    try
       FDataLink.Edit;
       FDataLink.UpdateRecord;
    except
    end;
end;

procedure TDBNumEdit.UpdateData(Sender : TObject);
begin
       try
          FDataLink.Field.Text := Text;
       except
          begin
             FDataLink.Edit;
             FDataLink.Field.Text := Text;
          end;
       end;
end;

procedure TDBNumEdit.DataChange(Sender : TObject);
begin
      try
          Text := FDataLink.Field.Text;
      except
          Text := '0';
      end;
end;

constructor TDBNumEdit.Create(AOwner: TComponent);
begin
    FDataLink := TFieldDataLink.Create;
    inherited Create(AOwner);
    FDataLink.OnDataChange := DataChange;
    FDataLink.OnUpdateData := UpdateData;
end;

destructor TDBNumEdit.Destroy;
begin
    FDataLink.Free;
    inherited Destroy;
end;

function TDBNumEdit.GetDataSource : TDataSource;
begin
   Result := FDataLink.DataSource;
end;
```

```
procedure TDBNumEdit.SetDataSource(Value : TDataSource);
begin
   FDataLink.DataSource := Value;
end;

function TDBNumEdit.GetDataField : string;
begin
   Result := FDataLink.FieldName;
end;

procedure TDBNumEdit.SetDataField(Value : string);
begin
   FDataLink.FieldName := Value;
end;

procedure Register;
begin
  RegisterComponents('KnowHow', [TDBNumEdit]);
end;

end.
```

CRIANDO UMA PROPRIEDADE QUE PERMITA TRATAR A TECLA ENTER COMO TAB

Normalmente os usuários de aplicações desenvolvidas para o ambiente Windows, principalmente aqueles acostumados a utilizar aplicações desenvolvidas para o ambiente MS-DOS, preferem utilizar a tecla Enter em vez da tecla Tab, para alterar o componente que recebe o foco.

Como já deve ser do seu conhecimento, essa característica pode ser implementada incluindo-se o seguinte trecho de código no procedimento associado ao evento OnKeyPress do componente:

```
if key = #13 then
begin
    Key := #0;
    Perform(WM_NEXTDLGCTL,0,0);
end;
```

Com essa codificação, ao se pressionar a tecla Enter o foco será deslocado para o próximo componente (seguindo-se a ordem estabelecida para a propriedade TabOrder dos componentes).

Evidentemente, seria muito mais simples incluir essa característica em uma propriedade do componente. Neste exemplo, definiremos para o componente DBNumEdit uma propriedade chamada TabEnter, do tipo booleano, que se for igual a true (seu valor default) fará com que, ao se pressionar a tecla Enter, o próximo componente receba o foco.

Para implementar essa propriedade devemos redefinir da seguinte maneira a unit que define o componente:

```
unit DBNumEdit;

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, NumEdit, dbCtrls, db;

type
  TDBNumEdit = class(TNumEdit)
```

```
  private
    { Private declarations }
    FDataLink : TFieldDataLink;
    FTabEnter : boolean;
    function GetDataSource : TDataSource;
    procedure SetDataSource(Value : TDataSource);
    function GetDataField : string;
    procedure SetDataField(Value : string);
    procedure DataChange(Sender : TObject);
    procedure UpdateData(Sender : TObject);
    procedure Change;override;
    procedure KeyPress(var Key : Char);override;
  protected
    { Protected declarations }
    procedure Notification(AComponent : TComponent;Operation : TOperation);override;
  public
    { Public declarations }
    constructor Create(AOwner: TComponent);override;
    destructor Destroy;override;
  published
    { Published declarations }
    property DataSource : TDataSource read GetDataSource write SetDataSource;
    property DataField : string read GetDataField write SetDataField;
    property TabEnter : boolean read FTabEnter write FTabEnter default True;
  end;

procedure Register;

implementation

procedure TDBNumEdit.KeyPress(var Key : Char);
Begin
     if (key = #13) and (TabEnter) then
     begin
        Key := #0;
        (Parent as TComponent).Perform(WM_NEXTDLGCTL,0,0);
     end
     else
     inherited KeyPress(Key);
end;

procedure TDBNumEdit.Notification(AComponent : TComponent;Operation : TOperation);
begin
     if (Operation = opRemove)and(AComponent is TDataSource)and(FDataLink.DataSource = nil)
     then FDataLink.FieldName := '';
end;

procedure TDBNumEdit.Change;
begin
    FDataLink.ModiFied;
    inherited Change;
    try
       FDataLink.Edit;
       FDataLink.UpdateRecord;
    except
    end;
end;

procedure TDBNumEdit.UpdateData(Sender : TObject);
begin
       try
          FDataLink.Field.Text := Text;
       except
          begin
```

```
              FDataLink.Edit;
              FDataLink.Field.Text := Text;
           end;
        end;
end;

procedure TDBNumEdit.DataChange(Sender : TObject);
begin
      try
          Text := FDataLink.Field.Text;
      except
          Text := '0';
      end;
end;

constructor TDBNumEdit.Create(AOwner: TComponent);
begin
    FDataLink := TFieldDataLink.Create;
    inherited Create(AOwner);
    FDataLink.OnDataChange := DataChange;
    FDataLink.OnUpdateData := UpdateData;
end;

destructor TDBNumEdit.Destroy;
begin
    FDataLink.Free;
    inherited Destroy;
end;

function TDBNumEdit.GetDataSource : TDataSource;
begin
   Result := FDataLink.DataSource;
end;

procedure TDBNumEdit.SetDataSource(Value : TDataSource);
begin
   FDataLink.DataSource := Value;
end;

function TDBNumEdit.GetDataField : string;
begin
   Result := FDataLink.FieldName;
end;

procedure TDBNumEdit.SetDataField(Value : string);
begin
   FDataLink.FieldName := Value;
end;

procedure Register;
begin
  RegisterComponents('KnowHow', [TDBNumEdit]);
end;

end.
```

Repare que, como queremos que o método Perform seja executado no formulário (ou objeto) que contém o componente, o código correto é:

```
(Parent as TComponent).Perform(WM_NEXTDLGCTL,0,0);
```

e não simplesmente

```
Perform(WM_NEXTDLGCTL,0,0);
```

Esta última linha de código não surtiria nenhum efeito.

Repare ainda que foi necessário sobrecarregar o método KeyPress da classe-base.

Esta última propriedade poderia ser definida na classe TNumEdit. Neste caso, estaria também presente na classe TDBNumEdit, derivada por herança da classe TNumEdit.

KNOW-HOW EM: CRIAÇÃO DE CONTROLES ACTIVEX

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 7.
- Experiência prévia na criação de componentes com o Delphi 7.

METODOLOGIA

- Apresentação do problema: Criação de controles ActiveX a partir de componentes desenvolvidos em Delphi 7.

TÉCNICA

- Apresentação dos procedimentos necessários à criação de Controles ActiveX.

APRESENTAÇÃO DO PROBLEMA

Conforme descrito nos tópicos precedentes, a criação de componentes usando-se a linguagem Object Pascal não oferece maiores dificuldades.

Considere agora a seguinte situação: você precisa desenvolver uma aplicação usando uma outra ferramenta de desenvolvimento que não o Delphi 7, e nessa aplicação precisa utilizar um componente (não disponível no ambiente) cujas características sejam idênticas às existentes em um componente desenvolvido em Object Pascal.

Se esse ambiente de desenvolvimento manipular controles ActiveX, e entre as características desejadas para o componente não se incluírem aquelas relacionadas à manipulação de bancos de dados, a sua tarefa será bastante simples, pois o Delphi 7 permite que se converta, de forma simples e rápida, um componente Delphi em um controle ActiveX (desde que esse componente não seja "data-aware").

A título de exemplificação, apresentaremos nos tópicos subseqüentes os procedimentos necessários à conversão do componente NumEdit em um controle ActiveX.

CONVERTENDO O COMPONENTE NUMEDIT EM UM CONTROLE ACTIVEX

Para transformar o componente NumEdit em um controle ActiveX, você deve executar os seguintes procedimentos:

1. Selecione o item New do menu File para exibir a caixa de diálogo New Items.

2. Selecione o item ActiveX Control da página ActiveX dessa caixa de diálogo, como mostrado na figura a seguir.

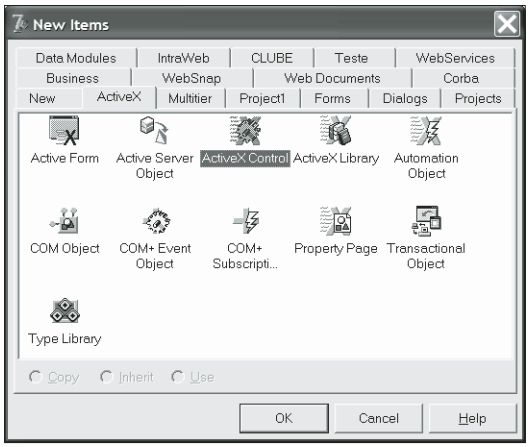


Figura 21.8: A caixa de diálogo New Items.

3. Selecione o botão Ok, para fechar essa caixa de diálogo e exibir a caixa de diálogo ActiveX Control Wizard, mostrada na figura a seguir.

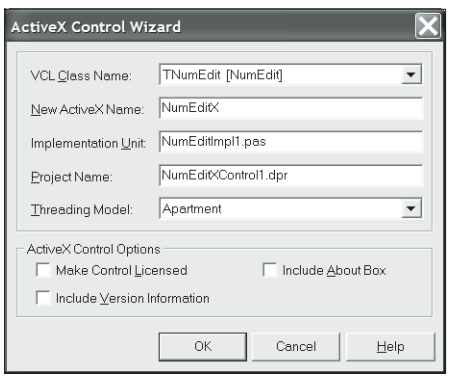


Figura 21.9: A caixa de diálogo ActiveX Control Wizard.

Nessa caixa de diálogo, devem-se definir:

- O nome da classe que define o componente a partir do qual será gerado o controle ActiveX.
- O nome do controle ActiveX que será gerado.
- O nome da unit na qual será implementado o controle.
- O nome do projeto responsável pela geração do controle ActiveX.
- O modelo de Thread empregado.

4. Selecione o botão Ok, para fechar essa caixa de diálogo. Caso o projeto corrente não corresponda a uma biblioteca ActiveX, será exibida a mensagem de advertência mostrada na figura a seguir. Selecione o botão Ok para gerar o novo projeto.

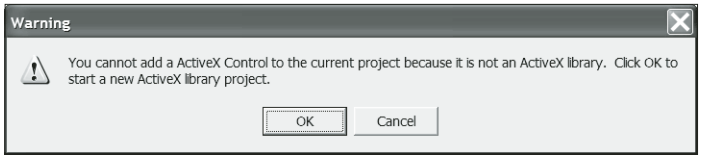


Figura 21.10: A mensagem de advertência.

Será então criada a unit responsável pela geração do controle ActiveX, cujo código é reproduzido a seguir.

```
unit NumEditImpl1;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, ActiveX, Classes, Controls, Graphics, Menus, Forms, StdCtrls,
  ComServ, StdVCL, AXCtrls, NumEditXControl1_TLB, NumEdit;

type
  TNumEditX = class(TActiveXControl, INumEditX)
  private
    { Private declarations }
    FDelphiControl: TNumEdit;
    FEvents: INumEditXEvents;
    procedure ChangeEvent(Sender: TObject);
    procedure ClickEvent(Sender: TObject);
    procedure DblClickEvent(Sender: TObject);
    procedure ErroConversaoEvent(Sender: TObject);
    procedure KeyPressEvent(Sender: TObject; var Key: Char);
  protected
    { Protected declarations }
    procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage); override;
    procedure EventSinkChanged(const EventSink: IUnknown); override;
    procedure InitializeControl; override;
    function DrawTextBiDiModeFlagsReadingOnly: Integer; safecall;
    function Get_AlignDisabled: WordBool; safecall;
    function Get_AutoSelect: WordBool; safecall;
    function Get_AutoSize: WordBool; safecall;
    function Get_BevelInner: TxBevelCut; safecall;
    function Get_BevelKind: TxBevelKind; safecall;
    function Get_BevelOuter: TxBevelCut; safecall;
    function Get_BorderStyle: TxBorderStyle; safecall;
    function Get_CanUndo: WordBool; safecall;
    function Get_CharCase: TxEditCharCase; safecall;
    function Get_Color: OLE_COLOR; safecall;
    function Get_Ctl3D: WordBool; safecall;
    function Get_DoubleBuffered: WordBool; safecall;
    function Get_DragCursor: Smallint; safecall;
    function Get_DragMode: TxDragMode; safecall;
    function Get_Enabled: WordBool; safecall;
    function Get_Font: IFontDisp; safecall;
    function Get_HideSelection: WordBool; safecall;
    function Get_ImeMode: TxImeMode; safecall;
    function Get_ImeName: WideString; safecall;
    function Get_MaxLength: Integer; safecall;
    function Get_Modified: WordBool; safecall;
    function Get_Numerico: WordBool; safecall;
    function Get_OEMConvert: WordBool; safecall;
    function Get_ParentColor: WordBool; safecall;
    function Get_ParentCtl3D: WordBool; safecall;
```

```
    function Get_PasswordChar: Smallint; safecall;
    function Get_ReadOnly: WordBool; safecall;
    function Get_SelLength: Integer; safecall;
    function Get_SelStart: Integer; safecall;
    function Get_SelText: WideString; safecall;
    function Get_Text: WideString; safecall;
    function Get_Visible: WordBool; safecall;
    function Get_VisibleDockClientCount: Integer; safecall;
    function IsRightToLeft: WordBool; safecall;
    function UseRightToLeftReading: WordBool; safecall;
    function UseRightToLeftScrollBar: WordBool; safecall;
    procedure _Set_Font(var Value: IFontDisp); safecall;
    procedure Clear; safecall;
    procedure ClearSelection; safecall;
    procedure ClearUndo; safecall;
    procedure CopyToClipboard; safecall;
    procedure CutToClipboard; safecall;
    procedure InitiateAction; safecall;
    procedure PasteFromClipboard; safecall;
    procedure SelectAll; safecall;
    procedure Set_AutoSelect(Value: WordBool); safecall;
    procedure Set_AutoSize(Value: WordBool); safecall;
    procedure Set_BevelInner(Value: TxBevelCut); safecall;
    procedure Set_BevelKind(Value: TxBevelKind); safecall;
    procedure Set_BevelOuter(Value: TxBevelCut); safecall;
    procedure Set_BorderStyle(Value: TxBorderStyle); safecall;
    procedure Set_CharCase(Value: TxEditCharCase); safecall;
    procedure Set_Color(Value: OLE_COLOR); safecall;
    procedure Set_Ctl3D(Value: WordBool); safecall;
    procedure Set_DoubleBuffered(Value: WordBool); safecall;
    procedure Set_DragCursor(Value: Smallint); safecall;
    procedure Set_DragMode(Value: TxDragMode); safecall;
    procedure Set_Enabled(Value: WordBool); safecall;
    procedure Set_Font(const Value: IFontDisp); safecall;
    procedure Set_HideSelection(Value: WordBool); safecall;
    procedure Set_ImeMode(Value: TxImeMode); safecall;
    procedure Set_ImeName(const Value: WideString); safecall;
    procedure Set_MaxLength(Value: Integer); safecall;
    procedure Set_Modified(Value: WordBool); safecall;
    procedure Set_Numerico(Value: WordBool); safecall;
    procedure Set_OEMConvert(Value: WordBool); safecall;
    procedure Set_ParentColor(Value: WordBool); safecall;
    procedure Set_ParentCtl3D(Value: WordBool); safecall;
    procedure Set_PasswordChar(Value: Smallint); safecall;
    procedure Set_ReadOnly(Value: WordBool); safecall;
    procedure Set_SelLength(Value: Integer); safecall;
    procedure Set_SelStart(Value: Integer); safecall;
    procedure Set_SelText(const Value: WideString); safecall;
    procedure Set_Text(const Value: WideString); safecall;
    procedure Set_Visible(Value: WordBool); safecall;
    procedure SetSubComponent(IsSubComponent: WordBool); safecall;
    procedure Undo; safecall;
  end;

implementation

uses ComObj;

{ TNumEditX }

procedure TNumEditX.DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage);
begin
  {TODO: Define property pages here.  Property pages are defined by calling
    DefinePropertyPage with the class id of the page.  For example,
```

```
      DefinePropertyPage(Class_NumEditXPage); }
end;

procedure TNumEditX.EventSinkChanged(const EventSink: IUnknown);
begin
  FEvents := EventSink as INumEditXEvents;
end;

procedure TNumEditX.InitializeControl;
begin
  FDelphiControl := Control as TNumEdit;
  FDelphiControl.OnChange := ChangeEvent;
  FDelphiControl.OnClick := ClickEvent;
  FDelphiControl.OnDblClick := DblClickEvent;
  FDelphiControl.OnErroConversao := ErroConversaoEvent;
  FDelphiControl.OnKeyPress := KeyPressEvent;
end;

function TNumEditX.DrawTextBiDiModeFlagsReadingOnly: Integer;
begin
  Result := FDelphiControl.DrawTextBiDiModeFlagsReadingOnly;
end;

function TNumEditX.Get_AlignDisabled: WordBool;
begin
  Result := FDelphiControl.AlignDisabled;
end;

function TNumEditX.Get_AutoSelect: WordBool;
begin
  Result := FDelphiControl.AutoSelect;
end;

function TNumEditX.Get_AutoSize: WordBool;
begin
  Result := FDelphiControl.AutoSize;
end;

function TNumEditX.Get_BevelInner: TxBevelCut;
begin
  Result := Ord(FDelphiControl.BevelInner);
end;

function TNumEditX.Get_BevelKind: TxBevelKind;
begin
  Result := Ord(FDelphiControl.BevelKind);
end;

function TNumEditX.Get_BevelOuter: TxBevelCut;
begin
  Result := Ord(FDelphiControl.BevelOuter);
end;

function TNumEditX.Get_BorderStyle: TxBorderStyle;
begin
  Result := Ord(FDelphiControl.BorderStyle);
end;

function TNumEditX.Get_CanUndo: WordBool;
begin
  Result := FDelphiControl.CanUndo;
end;

function TNumEditX.Get_CharCase: TxEditCharCase;
```

```
begin
  Result := Ord(FDelphiControl.CharCase);
end;

function TNumEditX.Get_Color: OLE_COLOR;
begin
  Result := OLE_COLOR(FDelphiControl.Color);
end;

function TNumEditX.Get_Ctl3D: WordBool;
begin
  Result := FDelphiControl.Ctl3D;
end;

function TNumEditX.Get_DoubleBuffered: WordBool;
begin
  Result := FDelphiControl.DoubleBuffered;
end;

function TNumEditX.Get_DragCursor: Smallint;
begin
  Result := Smallint(FDelphiControl.DragCursor);
end;

function TNumEditX.Get_DragMode: TxDragMode;
begin
  Result := Ord(FDelphiControl.DragMode);
end;

function TNumEditX.Get_Enabled: WordBool;
begin
  Result := FDelphiControl.Enabled;
end;

function TNumEditX.Get_Font: IFontDisp;
begin
  GetOleFont(FDelphiControl.Font, Result);
end;

function TNumEditX.Get_HideSelection: WordBool;
begin
  Result := FDelphiControl.HideSelection;
end;

function TNumEditX.Get_ImeMode: TxImeMode;
begin
  Result := Ord(FDelphiControl.ImeMode);
end;

function TNumEditX.Get_ImeName: WideString;
begin
  Result := WideString(FDelphiControl.ImeName);
end;

function TNumEditX.Get_MaxLength: Integer;
begin
  Result := FDelphiControl.MaxLength;
end;

function TNumEditX.Get_Modified: WordBool;
begin
  Result := FDelphiControl.Modified;
end;
```

```
function TNumEditX.Get_Numerico: WordBool;
begin
  Result := FDelphiControl.Numerico;
end;

function TNumEditX.Get_OEMConvert: WordBool;
begin
  Result := FDelphiControl.OEMConvert;
end;

function TNumEditX.Get_ParentColor: WordBool;
begin
  Result := FDelphiControl.ParentColor;
end;

function TNumEditX.Get_ParentCtl3D: WordBool;
begin
  Result := FDelphiControl.ParentCtl3D;
end;

function TNumEditX.Get_PasswordChar: Smallint;
begin
  Result := Smallint(FDelphiControl.PasswordChar);
end;

function TNumEditX.Get_ReadOnly: WordBool;
begin
  Result := FDelphiControl.ReadOnly;
end;

function TNumEditX.Get_SelLength: Integer;
begin
  Result := FDelphiControl.SelLength;
end;

function TNumEditX.Get_SelStart: Integer;
begin
  Result := FDelphiControl.SelStart;
end;

function TNumEditX.Get_SelText: WideString;
begin
  Result := WideString(FDelphiControl.SelText);
end;

function TNumEditX.Get_Text: WideString;
begin
  Result := WideString(FDelphiControl.Text);
end;

function TNumEditX.Get_Visible: WordBool;
begin
  Result := FDelphiControl.Visible;
end;

function TNumEditX.Get_VisibleDockClientCount: Integer;
begin
  Result := FDelphiControl.VisibleDockClientCount;
end;

function TNumEditX.IsRightToLeft: WordBool;
begin
  Result := FDelphiControl.IsRightToLeft;
end;
```

```
function TNumEditX.UseRightToLeftReading: WordBool;
begin
  Result := FDelphiControl.UseRightToLeftReading;
end;

function TNumEditX.UseRightToLeftScrollBar: WordBool;
begin
  Result := FDelphiControl.UseRightToLeftScrollBar;
end;

procedure TNumEditX._Set_Font(var Value: IFontDisp);
begin
  SetOleFont(FDelphiControl.Font, Value);
end;

procedure TNumEditX.ChangeEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnChange;
end;

procedure TNumEditX.Clear;
begin
  FDelphiControl.Clear;
end;

procedure TNumEditX.ClearSelection;
begin
  FDelphiControl.ClearSelection;
end;

procedure TNumEditX.ClearUndo;
begin
  FDelphiControl.ClearUndo;
end;

procedure TNumEditX.ClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnClick;
end;

procedure TNumEditX.CopyToClipboard;
begin
  FDelphiControl.CopyToClipboard;
end;

procedure TNumEditX.CutToClipboard;
begin
  FDelphiControl.CutToClipboard;
end;

procedure TNumEditX.DblClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDblClick;
end;

procedure TNumEditX.ErroConversaoEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnErroConversao;
end;

procedure TNumEditX.InitiateAction;
begin
  FDelphiControl.InitiateAction;
end;
```

```
procedure TNumEditX.KeyPressEvent(Sender: TObject; var Key: Char);
var
  TempKey: Smallint;
begin
  TempKey := Smallint(Key);
  if FEvents <> nil then FEvents.OnKeyPress(TempKey);
  Key := Char(TempKey);
end;

procedure TNumEditX.PasteFromClipboard;
begin
  FDelphiControl.PasteFromClipboard;
end;

procedure TNumEditX.SelectAll;
begin
  FDelphiControl.SelectAll;
end;

procedure TNumEditX.Set_AutoSelect(Value: WordBool);
begin
  FDelphiControl.AutoSelect := Value;
end;

procedure TNumEditX.Set_AutoSize(Value: WordBool);
begin
  FDelphiControl.AutoSize := Value;
end;

procedure TNumEditX.Set_BevelInner(Value: TxBevelCut);
begin
  FDelphiControl.BevelInner := TBevelCut(Value);
end;

procedure TNumEditX.Set_BevelKind(Value: TxBevelKind);
begin
  FDelphiControl.BevelKind := TBevelKind(Value);
end;

procedure TNumEditX.Set_BevelOuter(Value: TxBevelCut);
begin
  FDelphiControl.BevelOuter := TBevelCut(Value);
end;

procedure TNumEditX.Set_BorderStyle(Value: TxBorderStyle);
begin
  FDelphiControl.BorderStyle := TBorderStyle(Value);
end;

procedure TNumEditX.Set_CharCase(Value: TxEditCharCase);
begin
  FDelphiControl.CharCase := TEditCharCase(Value);
end;

procedure TNumEditX.Set_Color(Value: OLE_COLOR);
begin
  FDelphiControl.Color := TColor(Value);
end;

procedure TNumEditX.Set_Ctl3D(Value: WordBool);
begin
  FDelphiControl.Ctl3D := Value;
end;
```

```
procedure TNumEditX.Set_DoubleBuffered(Value: WordBool);
begin
  FDelphiControl.DoubleBuffered := Value;
end;

procedure TNumEditX.Set_DragCursor(Value: Smallint);
begin
  FDelphiControl.DragCursor := TCursor(Value);
end;

procedure TNumEditX.Set_DragMode(Value: TxDragMode);
begin
  FDelphiControl.DragMode := TDragMode(Value);
end;

procedure TNumEditX.Set_Enabled(Value: WordBool);
begin
  FDelphiControl.Enabled := Value;
end;

procedure TNumEditX.Set_Font(const Value: IFontDisp);
begin
  SetOleFont(FDelphiControl.Font, Value);
end;

procedure TNumEditX.Set_HideSelection(Value: WordBool);
begin
  FDelphiControl.HideSelection := Value;
end;

procedure TNumEditX.Set_ImeMode(Value: TxImeMode);
begin
  FDelphiControl.ImeMode := TImeMode(Value);
end;

procedure TNumEditX.Set_ImeName(const Value: WideString);
begin
  FDelphiControl.ImeName := TImeName(Value);
end;

procedure TNumEditX.Set_MaxLength(Value: Integer);
begin
  FDelphiControl.MaxLength := Value;
end;

procedure TNumEditX.Set_Modified(Value: WordBool);
begin
  FDelphiControl.Modified := Value;
end;

procedure TNumEditX.Set_Numerico(Value: WordBool);
begin
  FDelphiControl.Numerico := Value;
end;

procedure TNumEditX.Set_OEMConvert(Value: WordBool);
begin
  FDelphiControl.OEMConvert := Value;
end;

procedure TNumEditX.Set_ParentColor(Value: WordBool);
begin
  FDelphiControl.ParentColor := Value;
end;
```

```
procedure TNumEditX.Set_ParentCtl3D(Value: WordBool);
begin
  FDelphiControl.ParentCtl3D := Value;
end;

procedure TNumEditX.Set_PasswordChar(Value: Smallint);
begin
  FDelphiControl.PasswordChar := Char(Value);
end;

procedure TNumEditX.Set_ReadOnly(Value: WordBool);
begin
  FDelphiControl.ReadOnly := Value;
end;

procedure TNumEditX.Set_SelLength(Value: Integer);
begin
  FDelphiControl.SelLength := Value;
end;

procedure TNumEditX.Set_SelStart(Value: Integer);
begin
  FDelphiControl.SelStart := Value;
end;

procedure TNumEditX.Set_SelText(const Value: WideString);
begin
  FDelphiControl.SelText := String(Value);
end;

procedure TNumEditX.Set_Text(const Value: WideString);
begin
  FDelphiControl.Text := TCaption(Value);
end;

procedure TNumEditX.Set_Visible(Value: WordBool);
begin
  FDelphiControl.Visible := Value;
end;

procedure TNumEditX.SetSubComponent(IsSubComponent: WordBool);
begin
  FDelphiControl.SetSubComponent(IsSubComponent);
end;

procedure TNumEditX.Undo;
begin
  FDelphiControl.Undo;
end;

initialization
  TActiveXControlFactory.Create(
    ComServer,
    TNumEditX,
    TNumEdit,
    Class_NumEditX,
    1,
    '',
    0,
    tmApartment);
end.
```

O código do arquivo de projeto é reproduzido a seguir (podendo ser renomeado).

```
library NumEditXControl1;

uses
  ComServ,
  NumEditXControl1_TLB in 'NumEditXControl1_TLB.pas',
  NumEditImpl1 in 'NumEditImpl1.pas' {NumEditX: CoClass};

{$E ocx}

exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.TLB}

{$R *.RES}

begin
end.
```

5. Selecione o item Build All Projects (ou Build NumEditXControl1) do menu Project para gerar o arquivo NumEditXControl.ocx, no qual está definido o controle ActiveX.
6. Selecione o item Register ActiveX Server do menu Run, para registrar o controle no sistema operacional.

Caso o componente seja registrado com sucesso, será exibida a caixa de diálogo mostrada na figura a seguir.

Figura 21.11: Registrando o Controle Activex.

O controle gerado pode então ser usado por qualquer ambiente de desenvolvimento que suporte controles ActiveX, como o Microsoft Visual Basic 5.0, por exemplo.

A figura a seguir mostra a inclusão do controle no ambiente de desenvolvimento do Microsoft Visual Basic 6.0.

A Figura 21.12 mostra a janela de propriedades do Microsoft Visual Basic 6.0 para o controle recém-criado.

O controle ActiveX recém-criado apresenta as mesmas características definidas para o componente usado no Delphi.

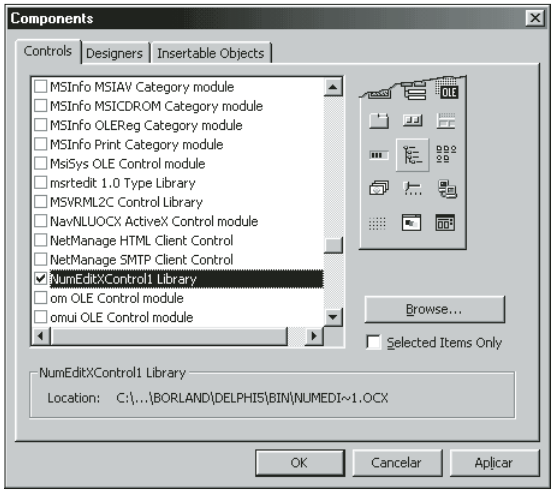


Figura 21.12: Inclusão do controle no ambiente de desenvolvimento do Microsoft Visual Basic 6.0.

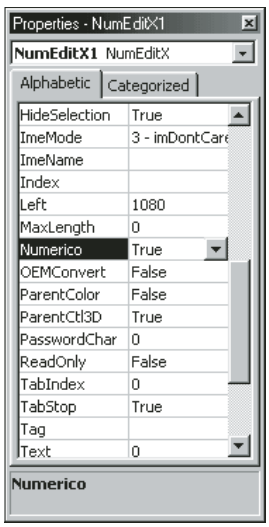
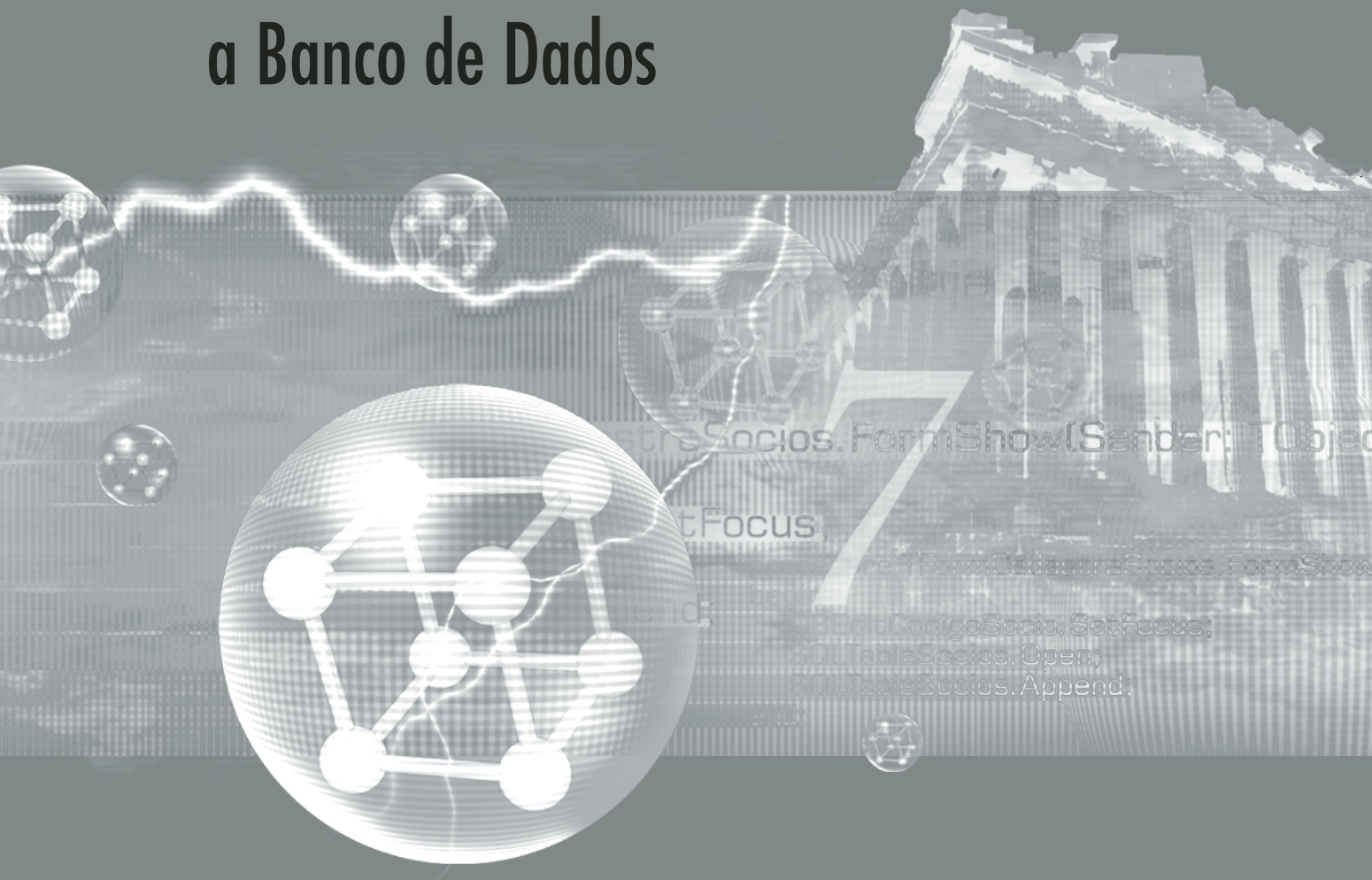


Figura 21.13: A janela de propriedades do Microsoft Visual Basic 6.0 para o controle recém-criado.

Capítulo 22

Mecanismos de Acesso a Banco de Dados



Neste capítulo serão apresentados tópicos gerais relacionados aos procedimentos necessários ao acesso a bancos de dados a partir de uma aplicação desenvolvida em Delphi 7, usando os mecanismos de acesso disponíveis.

KNOW-HOW em: Fundamentos dos Mecanismos de Acesso a Bancos de Dados

PRÉ-REQUISITOS

- Conhecimentos básicos relacionados à organização de informações.
- Conhecimentos básicos relacionados à utilização de componentes.

METODOLOGIA

- Apresentação dos conceitos fundamentais dos mecanismos de acesso a bancos de dados, e sua terminologia.

Os Mecanismos de Acesso a Bancos de Dados

Ao lançar a sua primeira ferramenta RAD para o desenvolvimento de aplicações para o ambente Windows (o Delphi 1.0) há alguns anos atrás, a Borland procurou apresentar ao mercado uma ferramenta que não simplificasse apenas a criação da interface com o usuário da aplicação, mas também a forma pela qual esta aplicação acessaria os diferentes formatos de bancos de dados existentes no mercado.

A Borland desejava que as aplicações criadas pudessem acessar diferentes bases de dados sem que fossem necessárias grandes alterações na codificação dos aplicativos criados, e isto foi possível graças à introdução de um mecanismo de acesso a bancos de dados também desenvolvido pela Borland – o Borland Database Engine (BDE).

Este mecanismo foi um dos grandes responsáveis pelo sucesso alcançado pelo Delphi como ferramenta de desenvolvimento (além de fatores mencionados em capítulos anteriores, como facilidade de criação de interface, orientação a objetos, etc.).

Apesar do grande sucesso alcançado pelo BDE, muitos desenvolvedores que utilizavam bancos de dados da Microsoft (como o Access e o SQL Server) reclamavam a ausência do suporte à tecnologia Activex Data Objects (ADO), presente na maioria dos computadores que executavam suas tarefas sobre o sistema operacional Windows.

Por esta razão o Delphi, a partir da sua versão 5, criou componentes com a finalidade específica de acessar bancos de dados via ADO, sem que fosse então necessária a instalação do BDE (supondo evidentemente que o suporte a ADO estivesse instalado nas máquinas onde a aplicação fosse ser executada – se não seria necessária a sua instalação).

Além disso, a versão 5 passava a incluir também um conjunto de componentes com a finalidade de permitir o desenvolvimento de aplicações cliente-servidor que obtivessem acesso nativo a bases de dados do Interbase sem a necessidade de instalação do BDE – conjunto este denominado Interbase Express.

A versão 6 do Delphi trouxe um novo mecanismo de acesso multiplataforma denominado DBExpress, mais leve do que o BDE, mas que só permite o acesso a sistemas gerenciadores de bancos de dados SQL, não oferecendo suporte a bancos de dados locais baseados em arquivos, como o Paradox e o DBase, por exemplo.

A versão 7 do Delphi trouxe um aprimoramento do DBExpress, e o "congelamento"do Borland Database Engine, para o qual não deverão ser oferecidas novas versões ou atualizações, ou suporte técnico.

Desta maneira, nosso estudo sobre bancos de dados será inicialmente subdividido nos seguintes capítulos:

- Desenvolvimento de Aplicações para acesso a Bancos de Dados via BDE.
- Desenvolvimento de Aplicações para acesso a Bancos de Dados via ADO.
- Desenvolvimento de Aplicações para acesso a Bancos de Dados via DBExpress.
- Desenvolvimento de Aplicações para acesso a Bancos de Dados via Interbase Express.

É importante caracterizar, no entanto, que a portabilidade de uma aplicação não fica comprometida com a existência de tantas opções, uma vez que a interface da aplicação poderá ser integralmente aproveitada ao se alternar de uma tecnologia para outra – sendo necessário apenas modificar os componentes de acesso à base de dados, geralmente situados em um repositório chamado Datamodule, cuja finalidade é justamente armazenar estes componentes, e alterar algumas poucas propriedades.

KNOW-HOW em: Classes Fundamentais de Acesso a Bancos de Dados – A Classe TDataset

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 7.
- Experiência prévia na utilização de classes e componentes no desenvolvimento de aplicações com Delphi 7.

METODOLOGIA

- Apresentação das classes e componentes fundamentais de acesso a bancos de dados, juntamente com uma descrição das suas propriedades, métodos e eventos.

TÉCNICA

- Descrição da classe TDataset, para acesso a bancos de dados.

Inicialmente será apresentada a classe TDataSet, a partir da qual são derivadas, por herança, as classes mais utilizadas para acessar os bancos de dados a serem manipulados por uma aplicação desenvolvida em Delphi, através dos diversos mecanismos de acesso.

A Classe TDataSet

Observando-se o diagrama de hierarquia de classes da VCL, verifica-se que, no topo das classes de acesso a bancos de dados, encontra-se a classe TDataSet. Dela são derivadas:

- A classe TTable, que representa uma tabela acessada via BDE.
- A classe TADOTable, que representa uma tabela acessada via ADO.
- A classe TIBTable, que representa uma tabela do Interbase acessada via Interbase Express.
- A classe TSQLTable, que representa uma tabela acessada via DBExpress.
- A classe TClientDataSet,que representa uma tabela acessada diretamente pela aplicação (sendo esta tabela armazenada no formato binário, no mesmo computador em que a aplicação é executada), ou em uma aplicação distribuída. Pode ser usada também para permitir, em conjunto com um componente DatasetProvider, acesso bidirecional (em memória) a bancos de dados acessados através de um mecanismo de acesso unidirecional.
- A classe TSQLSimpleDataSet, que permitie acesso bidirecional (em memória) a bancos de dados acessados através do mecanismo de acesso unidirecional DBExpress. Consiste em reunir, em uma única classe, objetos das classes TSQLDataset, DatasetProvider e ClientDataset, e substitui a classe TClientDataset da versão 6.
- A classe TQuery, que permite o acesso a uma ou mais tabelas via declarações SQL, através do Borland Database Engine.
- A classe TADOQuery, que permite o acesso a uma ou mais tabelas via declarações SQL, através da tecnologia ADO.
- A classe TIBQuery, que permite o acesso a uma ou mais tabelas via declarações SQL, através da tecnologia Interbase Express.
- A classe TSQLQuery, que permite o acesso a uma ou mais tabelas via declarações SQL, através da tecnologia DBExpress.
- A classe TStoredProc, que permite executar procedimentos armazenados em um banco de dados acessado através do Borland Database Emgine.
- A classe TADOStoredProc, que permite executar procedimentos armazenados em um banco de dados acessado através da tecnologia ADO.
- A classe TIBStoredProc, que permite executar procedimentos armazenados em um banco de dados acessado através da tecnologia Interbase Express.
- A classe TSQLStoredProc, que permite executar procedimentos armazenados em um banco de dados acessado através da tecnologia DBExpress.
- A classe TSQLDataset, que é uma generalização das classes TSQLTable, TSQLQuery e TSQLStoredProc, e que permite acessar diretamente uma tabela, uma ou várias tabelas via declarações SQL, ou executar procedimentos armazenados em um banco de dados acessado através da tecnologia DBExpress.

Um objeto da classe TDataSet, ou de qualquer das classes dela derivadas por herança, tem por objetivo representar tabelas a serem acessadas pela aplicação, sendo uma tabela definida, de forma genérica, por um conjunto de informações organizadas em linhas (registros da tabela) e colunas (campos da tabela). No caso de declarações SQL, também poderá ser retornado um conjunto de registros, ainda que provenientes de várias tabelas.

Dessa maneira, o acesso a tabelas definidas em um banco de dados será feito sempre através de um componente derivado da classe TDataSet.

A classe TDataSet, no entanto, implementa a funcionalidade genérica para acesso a tabelas, sem incorporar as funções da API do Borland Database Engine (o mecanismo de acesso a banco de dados criado pela Borland), os métodos da biblioteca ADO da Microsoft, do Interbase Express ou qualquer outro mecanismo de acesso. Estes métodos são implementados nas classes dela derivadas por herança.

Nos próximos tópicos, serão apresentadas as principais propriedades e métodos da classe TDataSet. Sempre que nos referirmos ao termo "Tabela", este deverá ser entendido como um conjunto de registros, ainda que retornados por uma declaração SQL.

Principais Propriedades da Classe TDataSet

Apresenta-se a seguir uma descrição das principais propriedades da classe TDataSet. Desta maneira, podemos descrever estas propriedades de uma única vez, ao invés de falar sobre elas cada vez que abordarmos uma das classes derivadas da classe TDataSet.

Active

Essa propriedade é definida como uma variável booleana, e define se a tabela (ou conjunto de tabelas) associada ao componente está ou não ativa.

Conforme será visto posteriormente, atribuir o valor true a essa propriedade equivale a executar o método Open do componente. Da mesma forma, atribuir o valor false a essa propriedade equivale a executar o método Close do componente.

AutoCalcFields

Essa propriedade é definida como uma variável booleana, e define se o procedimento associado ao evento OnCalcFields do componente deve ser executado sempre que houver uma alteração nos dados armazenados na tabela representada pelo componente.

Os campos calculados são aqueles cujo valor pode ser diretamente obtido em função dos valores armazenados em outros campos da tabela, razão pela qual não precisam ser armazenados permanentemente na tabela.

A definição da fórmula utilizada para o cálculo dos valores desses campos é feita no procedimento associado ao evento OnCalcFields do componente, conforme será exemplificado posteriormente.

BOF

Essa propriedade é definida como uma variável booleana, e define se o registro corrente é o primeiro registro da tabela representada pelo componente. É uma propriedade apenas de leitura, e tem o valor true em cada uma das seguintes situações:

- Quando se estabelece o acesso à tabela representada pelo componente, definido-se como True o valor da sua propriedade Active, ou após uma chamada ao seu método Open.

- Após uma chamada ao método First do componente.
- Após uma chamada ao método Prior do componente, quando o registro corrente já é o primeiro registro da tabela.

EOF

Essa propriedade é definida como uma variável booleana, e define se o registro corrente é o último registro da tabela representada pelo componente. É uma propriedade apenas de leitura, e tem o valor True em cada uma das seguintes situações:

- Quando se estabelece o acesso à tabela representada pelo componente, definido-se como True o valor da sua propriedade Active (ou após uma chamada ao seu método Open) e a tabela não possui nenhum registro armazenado.
- Após uma chamada ao método Last do componente.
- Após uma chamada ao método Next do componente, quando o registro corrente já é o último registro da tabela.

FieldCount

Essa propriedade é definida como uma variável inteira, e define o número de campos da tabela representada pelo componente. É uma propriedade apenas de leitura, e não pode ter o seu valor diretamente alterado pelo usuário.

É importante salientar que essa propriedade retorna o número de campos acessado pelo componente – que não é necessariamente igual ao número de campos reais da tabela representada pelo componente. Uma forma de se alterar o número de campos acessados pela tabela consiste em utilizar o Fields Editor, que permite que se definam os campos da tabela que serão realmente acessados pelo componente.

Fields

Essa propriedade é definida como uma array de objetos da classe TField, e permite o acesso a campos individuais do registro corrente da tabela representada pelo componente.

Nesse caso, o primeiro campo é representado pelo índice 0, sendo o n-ézimo campo representado pelo índice n-1.

É importante lembrar que esses campos são aqueles realmente acessados pelo componente, e não são necessariamente todos os campos da tabela por ele representada.

Para acessar os valores armazenados em cada campo acessado pelo componente, deve-se utilizar uma das propriedades de conversão da classe TField, dentre as quais podem-se destacar:

- AsString, para tratar o valor armazenado no campo como uma string.
- AsInteger, para tratar o valor armazenado no campo como um número inteiro.
- AsFloat, para tratar o valor armazenado no campo como um número real.

- AsBoolean, para tratar o valor armazenado no campo como um valor booleano.
- AsCurrency, para tratar o valor armazenado no campo como um valor monetário.
- AsDateTime, para tratar o valor armazenado no campo como um valor no formato Data/Hora.
- AsVariant, para tratar o valor armazenado no campo como um valor do Variant (que pode armazenar qualquer tipo de dado).

Dessa maneira, se quisermos exibir o valor armazenado no décimo campo acessado pelo componente, devemos empregar uma linha de código com a seguinte sintaxe:

```
ShowMessage(Table1.Fields[9].AsString);
```

Nesse caso, Table1 é o nome do componente que representa a tabela.

FieldValues

Essa propriedade é definida como uma array de valores do tipo Variant, indexado pelos nomes dos campos da tabela representada pelo componente. É interessante destacar que essa é a propriedade default da classe TDataSet e das classes dela derivadas por herança. Conseqüentemente, se TblPais é o nome de um componente de uma classe derivada de TDataSet, as linhas de código a seguir são equivalentes.

```
TblPais['Nome'] := Brasil;
TblPais.FieldValues['Nome'] := Brasil;
```

Filter

Essa propriedade é definida como uma variável do tipo string, e permite que se estabeleça uma condição (filtro) a ser atendida pelos registros da tabela acessada por esse componente.

Para que essa condição seja aplicada, deve-se, no entanto, definir como True o valor da propriedade Filtered do componente.

Filtered

Essa propriedade é definida como uma variável booleana, e define se os registros provenientes da tabela representada pelo componente devem ou não atender a uma condição (filtro) especificada na sua propriedade Filter (definida anteriormente) ou por um código inserido no procedimento associado ao seu evento OnFilterRecord.

FilterOptions

Essa propriedade é definida como uma variável do tipo TFilterOptions que é, por sua vez, um conjunto de elementos do tipo TFilterOption, sendo esses elementos definidos da seguinte maneira:

- foCaseInsensitive: Não diferencia letras maiúsculas e minúsculas na aplicação do filtro.
- foNoPartialCompare: Não permite que se utilize um asterisco para definir comparações parciais na aplicação de um filtro. Esse elemento deve ser incluído quando se quer tratar o asterisco (*) como um caractere do filtro.

Inicialmente, essa propriedade é definida como um conjunto vazio, isto é, por default nenhuma das condições anteriores é especificada.

RecNo

Essa propriedade é definida como uma variável inteira, e define o número do registro corrente, dentre todos os registros provenientes da tabela representada pelo componente. É uma propriedade apenas de leitura, e não pode ter o seu valor diretamente alterado pelo usuário, podendo ter o seu valor alterado em função da aplicação de um filtro.

RecordCount

Essa propriedade é definida como uma variável inteira, e define o número de registros provenientes da tabela representada pelo componente. É uma propriedade apenas de leitura, e não pode ter o seu valor diretamente alterado pelo usuário, podendo, no entanto, ter o seu valor alterado em função da aplicação de um filtro.

State

Essa propriedade é uma variável do tipo TDataSetState que define os estados que podem ser assumidos por uma tabela representada pelo componente. Essa propriedade só está disponível durante a execução do aplicativo, e, por ser uma propriedade apenas de leitura, não pode ter o seu valor diretamente alterado pelo usuário.

Essa propriedade pode assumir um dos valores descritos a seguir:

- dsInactive: A tabela está inativa, ou seja, sua propriedade Active é igual a false.
- dsBrowse: A tabela está sendo consultada. Os registros podem ser visualizados, mas não podem ser alterados.
- dsEdit: A tabela está sendo editada.
- dsFilter: O procedimento associado ao evento OnFilterRecord está sendo executado.
- dsInsert: A tabela está em modo de Inserção (um novo registro foi incluído na tabela).
- dsSetKey: A tabela está sendo reindexada (aplica-se apenas a objetos da classe TTable, derivada por herança da classe TDataSet).
- dsCalcFields: O procedimento associado ao evento OnCalcFields está sendo executado.
- dsNewValue: Estado temporário, utilizado para indicar que a propriedade NewValue de um objeto da classe TField está sendo acessada.
- dsOldValue: Estado temporário, utilizado para indicar que a propriedade OldValue de um objeto da classe TField está sendo acessada.
- dsCurValue: Estado temporário, utilizado para indicar que a propriedade CurValue de um objeto da classe TField está sendo acessada.

Principais Métodos da Classe TDataSet

Apresenta-se a seguir os principais métodos da classe TDataset, sendo a maioria desses métodos virtuais ou abstratos (implementados nas classes derivadas).

Append

Declaração

```
procedure Append;
```

Esse método adiciona um novo registro após o último registro existente na tabela representada pelo componente.

AppendRecord

Declaração

```
procedure AppendRecord(const Values: array of const);
```

Esse método adiciona e grava um novo registro após o último registro existente na tabela representada pelo componente, atribuindo aos campos do registro valores passados como parâmetros na chamada do método.

Cancel

Declaração

```
procedure Cancel; virtual;
```

Esse método cancela as alterações realizadas no registro corrente da tabela representada pelo componente, desde a última chamada ao seu método Post.

ClearFields

Declaração

```
procedure ClearFields;
```

Esse método apaga os valores armazenados nos campos do registro corrente da tabela representada pelo componente, desde que a tabela esteja em modo de edição ou inserção, gerando uma exceção em caso contrário.

Close

Declaração

```
procedure Close;
```

Esse método desfaz a conexão à tabela representada pelo componente, atribuindo o valor False à propriedade Active do componente, como mostra o trecho de código a seguir, extraído da unit db.pas.

```
procedure TDataSet.Close;
begin
    Active := False;
end;
```

Create

Declaração

```
constructor Create(AOwner: TComponent);
```

Esse método é o método construtor da classe, e recebe como parâmetro um objeto da classe TComponent, que indica o seu componente proprietário.

Delete

Declaração

```
procedure Delete;
```

Esse método remove o registro corrente da tabela representada pelo componente, definindo o próximo registro como registro atual.

Destroy

Declaração

```
destructor Destroy;
```

Esse método é o método destrutor da classe, devendo ser utilizado apenas quando uma instância da classe é criada dinamicamente pelo código da aplicação (mediante uma chamada ao método Create do componente que representa a tabela).

DisableControls

Declaração

```
procedure DisableControls;
```

Esse método desconecta temporariamente todos os componentes associados ao componente que representa a tabela.

Edit

Declaração

```
procedure Edit;
```

Esse método coloca a tabela representada pelo componente em modo de edição, permitindo que se alterem os valores armazenados nos campos do registro corrente.

EnableControls

Declaração

```
procedure EnableControls;
```

Esse método restabelece a conexão a todos os componentes associados ao componente que representa a tabela, cuja conexão tenha sido desabilitada por uma chamada ao método DisableControls.

FieldByName

Declaração

```
function FieldByName(const FieldName: string): TField;
```

Esse método retorna um objeto da classe TField que representa o campo cujo nome é passado como parâmetro na chamada ao método.

O valor retornado pelo método pode, evidentemente, ser tratado como um objeto da classe TField, e as propriedades de conversão descritas anteriormente podem ser utilizadas normalmente.

A utilização desse método apresenta, em relação à propriedade Fields, a vantagem de se poder desconsiderar a ordem exata dos campos, bastando que se conheça o nome de cada um deles.

FindField

Declaração

```
function FindField(const FieldName: string): TField;
```

Esse método determina se o componente acessa um campo cujo nome é passado como parâmetro na forma de uma string, retornando um objeto da classe TField que representa o campo caso o mesmo exista, e retornando nil em caso contrário.

First

Declaração

```
procedure First;
```

Esse método define o primeiro registro da tabela representada pelo componente como registro corrente. Além disso, antes de alterar o registro corrente grava qualquer alteração pendente, mediante uma chamada implícita ao método Post do componente.

Insert

Declaração

```
procedure Insert;
```

Esse método insere um novo registro na posição definida pelo registro corrente.

InsertRecord

Declaração

```
procedure InsertRecord(const Values: array of const);
```

Esse método insere e grava um novo registro na posição definida pelo registro corrente da tabela representada pelo componente, atribuindo aos campos do registro valores passados como parâmetros na chamada do método.

IsEmpty

Declaração

```
function IsEmpty: Boolean;
```

Esse método retorna True se a tabela representada pelo componente não tiver nenhum registro, retornando False em caso contrário.

IsLinkedTo

Declaração

```
function IsLinkedTo(DataSource: TDataSource): Boolean;
```

Esse método retorna True se o componente que representa a tabela estiver ligado a um componente DataSource, cujo nome é passado como parâmetro na chamada ao método, retornando False em caso contrário.

Last

Declaração

```
procedure Last;
```

Esse método define o último registro da tabela representada pelo componente como registro corrente. Além disso, antes de alterar o registro corrente grava qualquer alteração pendente, mediante uma chamada implícita ao método Post do componente.

Locate

Declaração

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean; virtual;
```

Esse método permite a busca exata de um registro, por campos que não façam parte do índice corrente da tabela representada pelo componente, e recebe como parâmetros:

- Uma string contendo os nomes dos campos pelos quais será feita a pesquisa (separados por ponto-e-vírgula).

- Uma string contendo os valores a serem pesquisados nos campos pelos quais será feita a pesquisa (separados por ponto-e-vírgula).
- Um conjunto de opções, que pode conter os seguintes elementos: LoCaseInsensitive – se esse elemento for incluído, letras maiúsculas e minúsculas serão tratadas indiferentemente; LoPartialKey – indica que a pesquisa será aproximada.

MoveBy

Declaração

```
function MoveBy(Distance: Integer): Integer;
```

Esse método redefine o registro atual, deslocando o "ponteiro ou cursor" que representa o registro atual da tabela representada pelo componente por um número de registros que é passado como único parâmetro na chamada do método.

Next

Declaração

```
procedure Next;
```

Esse método define o próximo registro da tabela representada pelo componente como registro corrente. Além disso, antes de alterar o registro corrente grava qualquer alteração pendente, mediante uma chamada implícita ao método Post do componente.

Na realidade, esse método faz uma chamada ao método MoveBy da classe, passando o valor 1 como parâmetro, como mostra o trecho de código reproduzido a seguir e extraído da unit db.pas.

```
procedure TDataSet.Next;
begin
  MoveBy(1);
end;
```

Open

Declaração

```
procedure Open;
```

Esse método estabelece a conexão à tabela representada pelo componente, atribuindo o valor True à sua propriedade Active, como mostra o trecho de código a seguir, extraído da unit db.pas.

```
procedure TDataSet.Open;
begin
  Active := True;
end;
```

Post

Declaração

```
procedure Post; virtual;
```

Esse método grava as alterações feitas no registro atual da tabela representada pelo componente. As alterações feitas anteriormente a uma chamada ao método Post não podem ser canceladas mediante uma chamada ao método Cancel (para que isso seja possível, deve-se empregar o conceito de transações, a ser descrito posteriormente).

Prior

Declaração

```
procedure Prior;
```

Esse método define o registro anterior da tabela representada pelo componente como registro corrente. Além disso, antes de alterar o registro corrente grava qualquer alteração pendente, mediante uma chamada implícita ao método Post do componente.

Na realidade, esse método faz uma chamada ao método MoveBy da classe, passando o valor -1 como parâmetro, como mostra o trecho de código a seguir, extraído da unit db.pas.

```
procedure TDataSet.Prior;
begin
  MoveBy(-1);
end;
```

Refresh

Declaração

```
procedure Refresh;
```

Esse método atualiza a exibição dos dados armazenados em uma tabela nos componentes de visualização conectados ao componente que representa a tabela. Uma situação em que deve ser feita uma chamada ao método refresh do componente corresponde àquela em que a aplicação de um filtro é cancelada (modificando-se o valor da sua propriedade Filtered para False).

SetFields

Declaração

```
procedure SetFields(const Values: array of const);
```

Esse método permite que se atribuam simultaneamente valores a vários campos do registro corrente da tabela representada pelo componente. Os valores devem ser fornecidos na ordem em que os campos são armazenados na tabela.

Para executar esse método a tabela deve estar em modo de edição, o que pode ser garantido por uma chamada ao método Edit do componente que a representa.

Principais Eventos da Classe TDataSet

Apresenta-se a seguir uma descrição dos principais eventos da classe TDataSet.

AfterCancel

O procedimento associado a esse evento é executado imediatamente após se cancelarem as alterações feitas no registro corrente, normalmente após uma chamada ao método Cancel do componente que representa a tabela.

Na realidade, o método Cancel faz uma chamada a um método protegido da classe TDataSet, denominado DoAfterCancel, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

AfterClose

O procedimento associado a esse evento é executado imediatamente após se interromper a conexão estabelecida entre a tabela e o componente que a representa, normalmente após uma chamada ao seu método Close, ou atribuindo-se o valor False à sua propriedade Active.

Na realidade, o método Close, ao atribuir o valor False à propriedade Active, provoca uma chamada a um método protegido da classe TDataSet, denominado DoAfterClose, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa. O mesmo é feito pelo método interno SetActive, responsável pela atribuição de um valor ao campo que armazena internamente o valor da propriedade Active.

AfterDelete

O procedimento associado a esse evento é executado imediatamente após se remover o registro corrente, normalmente após uma chamada ao método Delete do componente que representa a tabela.

Na realidade, o método Delete faz uma chamada a um método protegido da classe TDataSet, denominado DoAfterDelete, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

AfterEdit

O procedimento associado a esse evento é executado imediatamente após se colocar a tabela representada pelo componente em modo de edição.

Na realidade, o método Edit faz uma chamada a um método protegido da classe TDataSet, denominado DoAfterEdit, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

AfterInsertt

O procedimento associado a esse evento é executado imediatamente após a inserção de um novo registro na tabela representada pelo componente.

Na realidade, os métodos Insert e Append fazem uma chamada a um método protegido da classe TDataSet, denominado DoAfterInsertt, que verifica se existe um procedimento associado a esse evento e, em caso positivo, o executa.

AfterOpen

O procedimento associado a esse evento é executado imediatamente após se estabelecer a conexão entre a tabela e o componente que a representa, normalmente após uma chamada ao seu método Open, ou atribuindo-se o valor True à sua propriedade Active.

Na realidade, o método Open, ao atribuir o valor True à propriedade Active, faz uma chamada a um método protegido da classe TDataSet, denominado DoAfterOpen, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa. O mesmo é feito pelo método interno SetActive, responsável pela atribuição de um valor ao campo que armazena internamente o valor da propriedade Active.

AfterPost

O procedimento associado a esse evento é executado imediatamente após a gravação das alterações feitas no registro corrente da tabela representada pelo componente, normalmente após uma chamada ao seu método Post.

Na realidade, o método Post faz uma chamada a um método protegido da classe TDataSet, denominado DoAfterPost, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

AfterScroll

O procedimento associado a esse evento é executado imediatamente após uma movimentação para um outro registro da tabela (que passa a ser o registro corrente), normalmente após uma chamada aos seus métodos First, Prior, Next e Last, dentre outros.

Na realidade, esses métodos fazem uma chamada a um método protegido da classe TDataSet, denominado DoAfterScroll, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

BeforeCancel

O procedimento associado a esse evento é executado imediatamente antes de se cancelarem as alterações feitas no registro corrente, normalmente após uma chamada ao método Cancel do componente que representa a tabela.

Na realidade, o método Cancel faz uma chamada a um método protegido da classe TDataSet, denominado DoBeforeCancel, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

BeforeClose

O procedimento associado a esse evento é executado imediatamente antes de se interromper a conexão estabelecida entre a tabela e o componente que a representa, normalmente após uma chamada ao seu método Close, ou atribuindo-se o valor False à sua propriedade Active.

Na realidade, o método Close faz uma chamada a um método protegido da classe TDataSet, denominado DoBeforeClose, que verifica se existe um procedimento associado ao evento e, em caso positivo, o

executa. O mesmo é feito pelo método interno SetActive, responsável pela atribuição de um valor ao campo que armazena internamente o valor da propriedade Active.

BeforeDelete

O procedimento associado a esse evento é executado imediatamente antes de se remover o registro corrente, normalmente após uma chamada ao método Delete do componente que representa a tabela.

Na realidade, o método Delete faz uma chamada a um método protegido da classe TDataSet, denominado DoBeforeDelete, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

BeforeEdit

O procedimento associado a esse evento é executado imediatamente antes de se colocar a tabela representada pelo componente em modo de edição.

Na realidade, o método Edit faz uma chamada a um método protegido da classe TDataSet, denominado DoBeforeEdit, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

BeforeInsertt

O procedimento associado a esse evento é executado imediatamente antes da inserção de um novo registro na tabela representada pelo componente.

Na realidade, os métodos Insert e Append fazem uma chamada a um método protegido da classe TDataSet, denominado DoBeforeInsertt, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

BeforeOpen

O procedimento associado a esse evento é executado imediatamente antes de se estabelecer a conexão entre a tabela e o componente que a representa, normalmente após uma chamada ao seu método Open, ou atribuindo-se o valor True à sua propriedade Active.

Na realidade, o método Open faz uma chamada a um método protegido da classe TDataSet, denominado DoBeforeOpen, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa. O mesmo é feito pelo método interno SetActive, responsável pela atribuição de um valor ao campo que armazena internamente o valor da propriedade Active.

BeforePost

O procedimento associado a esse evento é executado imediatamente antes da gravação das alterações feitas no registro corrente da tabela representada pelo componente, normalmente após uma chamada ao seu método Post.

Na realidade, o método Post faz uma chamada a um método protegido da classe TDataSet, denominado DoBeforePost, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

BeforeScroll

O procedimento associado a esse evento é executado imediatamente antes de uma movimentação para um outro registro da tabela (que passa a ser o registro corrente), normalmente após uma chamada aos seus métodos First, Prior, Next e Last, dentre outros.

Na realidade, esses métodos fazem uma chamada a um método protegido da classe TDataSet, denominado DoBeforeScroll, que verifica se existe um procedimento associado ao evento e, em caso positivo, o executa.

OnCalcFields

O procedimento associado a esse evento é executado sempre que os valores dos campos calculados da tabela representada pelo componente precisam ser calculados.

OnDeleteError

O procedimento associado a esse evento é executado sempre que a tentativa de se remover um registro da tabela, em uma chamada ao seu método Delete, falha.

OnFilterRecord

O procedimento associado a esse evento é executado sempre que se aplica um filtro aos registros da tabela, devendonesse procedimento ser estabelecida a condição a ser atendida pelos registros. A aplicação de um filtro a uma tabela pode ser feita atribuindo-se o valor True à sua propriedade Filtered.

OnNewRecord

O procedimento associado a esse evento é executado sempre que se adiciona um novo registro à tabela representada pelo componente, mediante uma chamada ao método Append ou Insert, podendo ser utilizado para se definir valores default para os campos do registro recém-criado.

OnPostError

O procedimento associado a esse evento é executado sempre que a tentativa de se gravar um registro da tabela, em uma chamada ao seu método Post, falha.

Conforme será visto nos próximos capítulos, as classes de acesso a dados via BDE, ADO, DBExpress e Interbase Express não são derivadas diretamente da classe TDataSet, mas de outras classes dela derivadas, e que são:

- TCustomADODataSet, para acesso via ADO.
- TBDEDataSet e TDBDataset, para acesso via BDE.
- TIBCustomDataset, para acesso via Interbase Express.
- TCustomSQLDataSet, para acesso via DBExpress.

A classes TClientDataset e TSQLClientDataset são derivadas diretamente da classe TCustomClientDataset.

A Classe TCustomConnection

A classe TCustomConnection é derivada por herança direta da classe TComponent, sendo a classe-base de classes usada para fazer uma conexão a um banco de dados através do BDE (componente TDatabase), ADO (componente TADOConnection), InterbaseExpress (TIBDatabase) e DBExpress (TSQLConnection).

Principais Propriedades da Classe TCustomConnection

Apresenta-se a seguir uma descrição das principais propriedades da classe TCustomConnection, além daquelas herdadas das suas classes ancestrais.

Connected

Essa propriedade é definida como uma variável booleana e define se a conexão foi estabelecida.

DatasetCount

Essa propriedade é definida como uma variável inteira e define o número de componentes derivados da clase TDataset linkados a este componente de conexão.

Datasets

Essa propriedade é uma array de objetos de classes derivadas da classe TDataset, e permite acessar os componentes linkados a este componente de conexão.

LoginPrompt

Essa propriedade é definida como uma variável booleana e define se uma caixa de diálogo deverá ser exibida quando se estabelece uma conexão.

StreamedConnected

Essa propriedade é definida como uma variável booleana e define se a conexão estava ativa quando o componente responsável pela conexão foi carregado a partir de um stream.

Principais Métodos da Classe TCustomConnection

Apresenta-se a seguir uma descrição dos principais métodos da classe TCustomConnection, além daqueles herdados das suas classes ancestrais.

Close

Declaração

```
procedure Close;
```

Esse método encerra uma conexão (equivale a definir a propriedade Connected como False).

DoConnect

Declaração

```
procedure DoConnect; virtual;;
```

Esse método estabelece a conexão ao banco de dados (usado pela propriedade connected para estabelecer a conexão).

DoDisConnect

Declaração

```
procedure DoDisConnect; virtual;;
```

Esse método finaliza a comexão ao banco de dados (usado pela propriedade comnected para finalizar a conexão).

GetConnected

Declaração

```
function GetConnected: Boolean; virtual;
```

Esse método retorna o valor armazenado na propriedade Connected.

GetDataset

Declaração

```
function GetDataSet(Index:Integer): TDataSet; virtual;
```

Esse método retorna uma referência a um dos objetos derivados da classe TDataset vinculados a esta conexão, recebendo como parâmetro o índice que define o objeto na propriedade Datasets.

GetDatasetCount

Declaração

```
function GetDataSetCount: Integer; virtual;
```

Esse método retorna o valor armazenado na propriedade DatasetCount.

Loaded

Declaração

```
procedure Loaded; override;
```

Esse método carrega o componente de conexão após todos os outros componentes do formulário ou Datamodule terem sido carregados na memória.

DeleteIndex

Declaração

```
procedure DeleteIndex(const Name: string);
```

Esse método remove o índice cujo nome é passado como parâmetro na forma de uma string.

Open

Declaração

```
procedure Open;
```

Esse método inicializa ou estabelece uma conexão.

SetConnected

Declaração

```
procedure SetConnected(Value: Boolean); virtual;
```

Esse método altera o valor da propriedade Connected, em função do argumento passado como parâmetro na chamada do método.

Principais Eventos da Classe TCustomConnection

Apresenta-se a seguir uma descrição dos principais métodos da classe TCustomConnection, além daqueles herdados das suas classes ancestrais.

AfterConnect

Esse evento ocorre assim que uma conexão é estabelecida.

AfterDisconnect

Esse evento ocorre assim que uma conexão é encerrada.

BeforeConnect

Esse evento ocorre antes que uma conexão seja estabelecida.

BeforeDisconnect

Esse evento ocorre antes que uma conexão seja encerrada.

OnLogin

O procedimento associado a esse evento é executado sempre que se estabelece uma nova conexão ao banco de dados representado pelo componente, e o valor da sua propriedade LoginPrompt é igual a True.

Esse procedimento tem, entre seus parâmetros, um objeto da classe TStrings chamado LogimParams no qual devem ser fornecidos os valores dos parâmetros USERNAME e PASSWORD, para permitir o acesso ao banco de dados.

Caso se esteja utilizando a caixa de diálogo padrão de Login, não há necessidade de se codificar esse procedimento.

Capítulo 23

Banco de Dados – Componentes de Acesso via BDE

Neste capítulo serão apresentados as classes e os componentes responsáveis pelo acesso a dados via BDE, a partir de uma aplicação VCL desenvolvida com o Borland Delphi 7. É importante destacar, inicialmente, que este mecanismo de acesso só está disponível para aplicações baseadas na VCL (inexistindo para aplicações baseadas na CLX).

Este mecanismo está presente desde a primeira versão do Delphi, e já é enorme a base de aplicações desenvolvidas com bvase nesta tecnologia. Por estas e outras razões, o BDE continua presente nesta nova versão do Delphi, embora seu desenvolvimento esteja descontinuado.

KNOW-HOW em: Classes Fundamentais de Acesso a Bancos de Dados via BDE – As Classes TBDEDataset e TDBDataset

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 7.
- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi 7.
- Compreensão da classe TDataset.

METODOLOGIA

- Apresentação das classes e dos componentes de acesso a bancos de dados via BDE, juntamente com uma descrição das suas propriedades, métodos e eventos.

TÉCNICA

- Descrição das classes e dos componentes de acesso a bancos de dados e apresentação de exemplos de aplicação.

Inicialmente apresentaremos as classes derivadas de TDataset que implementam as principais funcionalidades do Borland Database Engine e depois os componentes efetivamente usados no desenvolvimento de aplicações.

Alguns autores apresentam uma vasta coletânea de funções da API do BDE. Como as classes e componentes definidas pelo Delphi possuem justamente a finalidade de evitar que o desenvolvedor precise manipular diretamente estas funções, só as citarei quando isto for realmente indispensável. Afinal de contas, se você gostasse de programar usando diretamente funções de API´s, provavelmente não estaria usando o Delphi como ferramenta de desenvolvimento, mas um compilador C e as funções da API do Windows.

A Classe TBDEDataSet

Conforme descrito anteriormente, a classe TDataSet implementa a funcionalidade genérica para acesso a tabelas, sem incorporar as funções da API do Borland Database Engine – o primeiro mecanismo de acesso a banco de dados criado pela Borland.

A classe TBDEDataSet, derivada por herança direta da classe TDataSet, incorpora a API do Borland DataBase Engine a alguns dos métodos declarados na classe TDataSet, sobrecarregando-os (no caso de métodos virtuais) ou implementando-os (no caso de métodos abstratos).

Além disso, redefine como published propriedades e eventos declarados na seção public da classe-base, tornando-os acessíveis pelo Object Inspector.

Os métodos implementados por essa classe não são normalmente usados no desenvolvimento de aplicações em Delphi, pois geralmente são utilizados componentes representados por classes derivadas por herança da classe TBDEDataSet.

Principais Propriedades da Classe TBDEDataSet

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TDataSet.

CachedUpdates

Essa propriedade é definida como uma variável booleana, e define se o recurso de cached updates será utilizado ou não pelo componente.

O recurso de cached updates permite que uma aplicação obtenha um conjunto de registros de uma base de dados e faça sua edição local (que pode abranger edição, inclusão ou remoção de registros) e posterior atualização das informações na base de dados.

Além de reduzir o tráfego de informações em redes, esse recurso permite o cancelamento de alterações feitas de forma indesejada.

Para habilitar o recurso de cached updates, basta definir como True o valor da propriedade CachedUpdates do componente que representa a tabela.

A atualização dos registros na base de dados é feita mediante uma chamada aos métodos ApplyUpdates e CommitUpdates do componente que representa a tabela. O cancelamento das alterações é feito mediante uma chamada ao método CancelUpdates desse mesmo componente.

Principais Métodos da Classe TBDEDataSet

Apresenta-se a seguir uma descrição dos principais métodos da classe TBDEDataSet, além daquelas herdadas das suas classes-base.

ApplyUpdates

Declaração

```
procedure ApplyUpdates;
```

Esse método armazena no banco de dados as alterações feitas localmente. Essas alterações, no entanto, só são efetivadas após uma chamada ao método Commit do componente TDatabase que representa o banco de dados, se este estiver sendo empregado.

CancelUpdates

Declaração

```
procedure CancelUpdates;
```

Esse método cancela as alterações feitas localmente, para um componente que representa uma tabela em que o recurso de Cached Updates está habilitado.

CommitUpdates

Declaração

```
procedure CommitUpdates;
```

Esse método limpa o buffer de alterações locais, para um componente que representa uma tabela em que o recurso de Cached Updates está habilitado.

Locate

Declaração

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean;
```

Esse método permite a busca de um registro por campos que não façam parte do índice corrente da tabela. Recebe como parâmetros:

- Uma string contendo os nomes dos campos pelos quais será feita a pesquisa (separados por ponto-e-vírgula).
- Uma array do tipo Variant contendo os valores a serem pesquisados nos campos pelos quais será feita a pesquisa (separados por ponto-e-vírgula).
- Um conjunto de opções, que pode conter os seguintes elementos:
  - LoCaseInsensitive: Se esse elemento for incluído, letras maiúsculas e minúsculas serão tratadas indiferentemente.
  - LoPartialKey: A pesquisa será aproximada.

A Classe TDBDataSet

A classe TDBDataSet é derivada por herança direta da classe TBDEDataSet, sendo a ancestral direta das classes que representam os componentes freqüentemente utilizados no acesso a tabelas definidas em um banco de dados via BDE – TTable, TQuery e TStoredProc (este utilizado nas aplicações que acessam bancos de dados que implementam a filosofia cliente-servidor).

Diferentemente da classe TBDEDataSet, esta implementa propriedades e métodos utilizados freqüentemente no desenvolvimento de aplicações para acesso a bancos de dados em Delphi, ainda que utilizando em componentes representados por classes dela derivadas.

Principais Propriedades da Classe TDBDataSet

Apresenta-se a seguir uma descrição das principais propriedades da classe TDBDataSet, além daquelas herdadas das suas classes-base.

Database

Essa propriedade é definida como um objeto da classe TDatabase, e permite o acesso às propriedades e métodos do componente que representa o banco de dados ao qual se está conectado.

É importante lembrar que, se nenhum componente TDataBase for explicitamente incluído na aplicação, uma instância dessa classe será gerada automaticamente pelo Delphi.

DatabaseName

Essa propriedade é definida como uma variável do tipo string, e define o nome do banco de dados ao qual o componente está conectado. Pode ser o nome de um alias definido no Borland Database Engine ou o nome definido na propriedade DataBaseName do componente Database associado e, no caso de tabelas locais do dBASE e Paradox, por exemplo, pode ser definida como o diretório no qual as tabelas estão armazenadas.

DBSession

Essa propriedade é definida como um objeto da classe TSession, e define o nome do componente Session associado ao componente TDatabase que representa o banco de dados ao qual se está conectado. Se o componente não estiver explicitamente conectado a um objeto TDatabase, este será criado dinamicamente e conectado ao objeto Session default, a ser criado dinamicamente pela aplicação.

SessionName

Essa propriedade é definida como uma variável do tipo string, e define o nome do objeto da classe TSession associado ao componente TDatabase que representa o banco de dados ao qual se está conectado. Se o componente não estiver explicitamente conectado a um objeto TDatabase, este será criado dinamicamente e conectado ao objeto Session default, a ser criado dinamicamente pela aplicação.

A Classe TDatabase

A classe TDatabase é derivada por herança direta da classe TComponent, sendo normalmente utilizada para se estabelecer uma conexão a um banco de dados.

Essa classe permite que se incorpore o conceito de transações no acesso a um banco de dados através de uma aplicação desenvolvida em Delphi, além de permitir a personalização das caixas de diálogo utilizadas para validar o acesso de um usuário ao banco de dados representado pelo componente.

Principais Propriedades da Classe TDatabase

Apresenta-se a seguir uma descrição das principais propriedades da classe TDatabase.

AliasName

Essa propriedade é definida como uma variável do tipo string, e define o nome do alias, definido no Borland Database Engine, que representa o banco de dados que está sendo acessado.

DATABASENAME

Essa propriedade é definida como uma variável do tipo string, e define o nome pelo qual o componente será referenciado nos componentes a ele associados (definido-se esse valor na propriedade DatabaseName desses componentes). Não confundir com a sua propriedade Name, que define o nome pelo qual o componente será referenciado no código-fonte da aplicação.

DATASETS

Essa propriedade é definida como um array de objetos da classe TDBDataSet, e que retorna um conjunto de objetos da classe TDBDataSet (ou de classes delas derivadas por herança) associados ao banco de dados representado por esse componente, e cuja conexão está ativa.

DIRECTORY

Essa propriedade é definida como uma variável do tipo string, e define o nome do diretório de trabalho para bancos de dados que manipulam tabelas no formato Paradox ou dBASE.

DRIVERNAME

Essa propriedade é definida como uma variável do tipo string, e define o nome de um driver do Borland Database Engine.

As propriedades DatabaseName e AliasName são mutuamente excludentes. A especificação de um valor para uma dessas propriedades anula o valor definido para a outra.

INTRANSACTION

Essa propriedade é definida como uma variável booleana, e define se existe uma transação em progresso (o conceito de transações será apresentado no próximo capítulo).

ISSQLBASED

Essa propriedade é definida como uma variável booleana, e define se a conexão está sendo feita através de um driver SQL, ou localmente a tabelas dos tipos Paradox e dBASE.

KEEPCONNECTION

Essa propriedade é definida como uma variável booleana, e define se a conexão ao banco de dados deve ser mantida, ainda que todas as conexões estabelecidas por componentes derivados da classe TDataSet tenham sido encerradas.

PARAMS

Essa propriedade é definida como um objeto da classe TStrings (lista de strings) e pode ser usada para armazenar as definições dos parâmetros que devem ser informados quando se estabelece uma conexão ao banco de dados representado pelo componente.

Session

Essa propriedade é definida como um objeto da classe TSession, e define o nome do objeto da classe TSession ao qual o componente está conectado. Se nenhum componente da classe TSession for explicitamente incluído na aplicação, será usado um objeto da classe TSession criado dinamicamente pela aplicação.

TransIsolation

Essa propriedade especifica o nível de isolamento das transações gerenciadas pelo BDE, determinando como várias transações independentes se relacionam. Essa propriedade pode assumir os valores descritos na tabela a seguir:

- tiDirtyRead: Permite a leitura de alterações feitas por outras transações simultâneas, mas ainda não confirmadas por uma chamada ao método Commit.
- TiReadCommitted: Permite apenas a leitura de alterações feitas por outras transações simultâneas, mas que já tenham sido confirmadas por uma chamada ao método Commit.

Este é o valor default dessa propriedade.

- TiRepeatableRead: A transação não enxerga qualquer alteração feita por outras transações que atuam simultaneamente. Esse nível de isolamento garante que, uma vez que uma transação leia um registro, essa visão do registro não se altere a menos que a própria transação efetue alterações no registro.

Principais Métodos do Componente Database

Os principais métodos do componente Database, e o significado de cada um deles, são apresentados a seguir.

ApplyUpdates

Declaração

```
procedure ApplyUpdates(const DataSets: array of TDBDataSet);
```

Esse método aplica as alterações pendentes em componentes derivados da classe TDataSet associados a esse componente (e passados como parâmetros na chamada ao método, na forma de um array de objetos da classe TDBDataSet), para os quais a propriedade CachedUpdates possui o valor True.

CloseDataSetss

Declaração

```
procedure CloseDatasets;
```

Esse método encerra as conexões estabelecidas em componentes derivados da classe TDataSet associados a esse componente.

Commit

Declaração

```
procedure Commit;
```

Esse método confirma a execução de todos os comandos iniciados após a última chamada ao método StartTransaction. Esse método só pode ser executado se uma transação estiver sendo processada, o que pode ser constatado verificando-se o valor da propriedade InTransaction. Após uma chamada ao método Commit, a propriedade InTransaction assume o valor False.

RollBack

Declaração

```
procedure RollBack;
```

Esse método cancela todos os comandos iniciados após a última chamada ao método StartTransaction. Esse método só pode ser executado se uma transação estiver sendo processada, o que pode ser constatado verificando-se o valor da propriedade InTransaction. Após uma chamada ao método RollBack, a propriedade InTransaction assume o valor False.

StartTransaction

Declaração

```
procedure StartTransaction;
```

Esse método inicia uma nova transação, e só pode ser executado se não houver uma transação sendo processada, isto é, após uma chamada ao método Commit ou RollBack. Após uma chamada ao método StartTransaction, a propriedade InTransaction assume o valor True.

A Classe TSession

A classe TSession é derivada por herança direta da classe TComponent, sendo normalmente utilizada para se estabelecer uma conexão a um banco de dados.

Essa classe permite que se gerenciem múltiplas conexões a bancos de dados a partir de uma única aplicação e que aplicações precisando acessar tabelas Paradox em diretórios distintos de uma rede ou compartilhado a tabelas utilizem vários componentes da classe TSession.

Principais Propriedades da Classe TSession

Apresenta-se a seguir uma descrição das principais propriedades da classe TSession.

Active

Essa propriedade é determinada como uma variável booleana, e define se a sessão está ativa. Atribuir um valor False a essa propriedade gera as seguintes conseqüências:

A propriedade Connected dos componentes Database associados ao componente passa a ter o valor False, bem como os componentes de classes derivadas de TDataSet associados a esses componentes da classe TDatabase.

AutoSessionName

Essa propriedade é determinada como uma variável booleana, e define se a sessão terá ou não um nome gerado automaticamente (e armazenado na sua propriedade SessionName).

DatabaseCount

Essa propriedade é determinada como uma variável inteira, e define o número de componentes da classe TDatabase associados ao componente.

Databases

Essa propriedade é definida como um array de objetos da classe TDatabase, e que retorna um conjunto de objetos da classe TDatabase associados ao componente, e cuja conexão está ativa.

KeepConnection

Essa propriedade é determinada como uma variável booleana, e define se a conexão ao banco de dados criado temporariamente pelo componente deve ser mantida, ainda que todas as conexões estabelecidas por componentes derivados da classe TDataSet a ele associados tenham sido encerradas.

NetFileDir

Essa propriedade é determinada como uma variável do tipo string, e define o nome do diretório que armazena o arquivo de controle de rede do BDE (arquivo PDOXUSRS.NET).

PrivateDir

Essa propriedade é determinada como uma variável do tipo string, e define o nome do diretório de trabalho para bancos de dados que manipulam tabelas temporárias geradas pelo Borland database Engine no formato Paradox ou dBASE.

SessionName

Essa propriedade é determinada como uma variável do tipo string, e define o nome pelo qual a sessão representada pelo componente será identificada nos componentes da classe TDatabase que estarão conectados ao componente.

SQLHourGlass

Essa propriedade é determinada como uma variável booleana, e define se o cursor do mouse deve apresentar o formato de uma ampulheta durante a execução de operações por parte do Borland Database Engine (BDE).

TraceFlags

Essa propriedade é uma variável do tipo TTraceFlags, e indica as operações do Borland Database Engine que devem ser monitoradas pelo utilitário SQL Monitor.

A propriedade TTraceFlags é um conjunto que pode conter os seguintes elementos:

- TfQPrepare: Monitora operações do tipo Prepare.
- TfQExecute: Monitora operações do tipo ExecSQL.
- TfError: Monitora mensagens de erro do servidor.
- TfStmt: Monitora todas as operações decorrentes da execução de declarações SQL.
- TfConnect: Monitora operações de conexão a bancos de dados.
- TfTransact: Monitora operações de transações, como as decorrentes de chamadas aos métodos StartTransaction, Commit e RollBack.
- TfBlob: Monitora operações com dados do tipo blob.
- TfMisc: Monitora operações que não sejam definidas pelos outros flags.
- TfVendor: Monitora chamadas das funções da API do servidor de banco de dados.
- TfDataIn: Monitora os dados recebidos do servidor.
- TfDataOut: Monitora os dados enviados ao servidor.

Principais Métodos da Classe TSession

Apresenta-se a seguir uma descrição dos principais métodos da classe TSession.

AddAlias

Declaração

```
procedure AddAlias(const Name, Driver: string; List: TStrings);
```

Esse método permite que se crie um alias durante a execução do aplicativo. Esse método recebe como parâmetros uma string que define o nome do alias que está sendo criado, o driver do SQL Links para o servidor de banco de dados para o qual o alias será criado e um objeto da classe TStrings (uma lista de strings) que define os parâmetros para configuração do alias no BDE (esses parâmetros variam de acordo com o banco de dados selecionado e podem ser visualizados no SQL Explorer, exibido quando você seleciona o item Explore do menu Database).

AddPassword

Declaração

```
procedure AddPassword(const Password: string);
```

Esse método permite que se adicione uma senha para acessar tabelas criptografadas do tipo Paradox, sendo esta senha passada como parâmetro na forma de uma string, durante a chamada ao método.

AddStandardAlias

Declaração

```
procedure AddStandardAlias(const Name, Path, DefaultDriver: string);
```

Esse método é semelhante ao método AddAlias, mas recebe três strings como parâmetros. A primeira define o alias que está sendo criado, a segunda o path a que o alias se refere e a terceira pode ser igual a um dos seguintes valores: 'Paradox', 'dBASE' ou 'ASCIIDRV'. Para este último parâmetro, uma string nula equivale a 'Paradox'.

Close

Declaração

```
procedure Close;
```

Esse método encerra a sessão e todas as conexões realizadas por componentes da classe TDatabase associados ao componente.

CloseDatabase

Declaração

```
procedure CloseDatabase(Database: TDatabase);
```

Esse método encerra a conexão realizada por um componente da classe TDatabase associado ao componente, e cujo nome é passado como parâmetro na chamada do método.

DeleteAlias

Declaração

```
procedure DeleteAlias(const Name: string);
```

Esse método faz justamente o inverso do método AddAlias, isto é, remove um alias do BDE. Esse método só tem um parâmetro, que é uma string contendo o nome do alias a ser removido.

FindDatabase

Declaração

```
function FindDatabase(const DatabaseName: string): TDatabase;
```

Esse método verifica se entre os componentes da classe TDatabase associados ao componente existe um componente cujo nome é igual ao da string passada como parâmetro na chamada desse método.

GetAliasNames

Declaração

```
procedure GetAliasNames(List: TStrings);
```

Esse método coloca, em um objeto da classe TStrings passado como parâmetro, os nomes de todos os alias configurados pelo BDE.

GetAliasParams

Declaração

```
procedure GetAliasParams(const AliasName: string; List: TStrings);
```

Esse método recebe como primeiro parâmetro o nome de um alias (na forma de uma string) e retorna, em um objeto da classe TStrings (que deve ser passado como segundo parâmetro), os parâmetros do alias cujo nome foi fornecido.

IsAlias

Declaração

```
function IsAlias(const Name: string): Boolean;
```

Esse método recebe uma string como parâmetro e determina se já existe um alias com o nome definido por essa string.

GetTableNames

Declaração

```
procedure GetTableNames(const DatabaseName, Pattern: string; Extensions, SystemTables:
Boolean; List: TStrings);
```

Esse método recebe como parâmetros:

- O nome do banco de dados (que pode ser um componente TDataBase) cujos nomes das tabelas se quer acessar.
- Uma string que define um delimitador para as tabelas (deve ser nulo, quando se quer obter os nomes de todas as tabelas).
- Uma constante booleana que define se as extensões dos nomes das tabelas devem ser exibidas.
- Uma outra constante booleana que define se os nomes das tabelas de sistema que definem a estrutura das tabelas do banco de dados também devem ser obtidos (essa constante deve ter o valor False para tabelas dos tipos Paradox e dBASE).
- Um objeto da classe TStrings no qual serão armazenados os nomes das tabelas.

Open

Declaração

```
procedure Open;
```

Esse método inicializa uma sessão e a define como sendo a sessão corrente.

OpenDatabase

Declaração

```
function OpenDatabase(const DatabaseName: string): TDatabase;
```

Esse método estabelece a conexão de um componente da classe TDatabase associado ao componente, e cujo nome é passado como parâmetro na chamada do método.

RemoveAllPassword

Declaração

```
procedure RemovePassword(const Password: string);
```

Esse método permite que se removam todas as senhas definidas para acessar tabelas criptografadas do tipo Paradox, estabelecidas anteriormente em chamadas ao método AddPassword.

RemovePassword

Declaração

```
procedure RemovePassword(const Password: string);
```

Esse método permite que se remova uma senha para acessar tabelas criptografadas do tipo Paradox, sendo esta senha passada como parâmetro na forma de uma string, durante a chamada ao método.

Eventos do Componente Session

Apresenta-se a seguir uma breve descrição dos eventos associados a esse componente.

OnPassword

O procedimento associado a esse evento é executado sempre que se tenta acessar um arquivo Paradox que requer uma validação do usuário.

OnStartUp

O procedimento associado a esse evento é executado sempre que a sessão representada pelo componente é ativada.

KNOW-HOW em: Classes de Acesso Direto a Bancos de Dados via BDE – As Classes TTable e TQuery

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 7.
- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi 7.
- Compreensão da classes TDataset, TBDEDataset e TDBDataset.

METODOLOGIA

- Apresentação das classes e dos componentes de acesso direto a bancos de dados via BDE, juntamente com uma descrição das suas propriedades, métodos e eventos.

TÉCNICA

- Descrição das classes e dos componentes de acesso direto a bancos de dados e apresentação de exemplos de aplicação.
- Serão apresentadas neste tópico as classes usadas para acesso direto a bancos de dados via BDE – as classes Ttable e TQuery.

A Classe TTable

A classe TTable é derivada por herança direta da classe TDBDataSet, sendo normalmente utilizada para se estabelecer uma conexão a uma tabela individual do banco de dados.

Principais Propriedades da Classe TTable

Apresenta-se a seguir uma descrição das principais propriedades da classe TTable, além daquelas herdadas das suas classes ancestrais.

CanModify

Essa propriedade é definida como uma variável booleana e define se a tabela pode ser acessada para edição, inserção e remoção de registros.

Exclusive

Essa propriedade é definida como uma variável booleana e define se a tabela será acessada em modo exclusivo pela aplicação. Se seu valor for igual a True, nenhuma outra aplicação poderá acessar a tabela, enquanto a mesma estiver sendo acessada pela aplicação corrente.

IndexDefs

Essa propriedade é um objeto da classe TindexDefs e define os índices da tabela.

IndexFieldCount

Essa propriedade é definida como uma variável inteira e define o número de campos que compõem o índice corrente.

IndexFieldNames

Essa propriedade é definida como uma variável do tipo string, que armazena os nomes dos campos que compõem o índice corrente, separados por um ponto-e-vírgula.

As propriedades IndexName e IndexFieldNames são mutuamente excludentes, isto é, a definição de um valor para uma propriedade anula o valor definido para a outra.

IndexFields

Essa propriedade é definida como um array de objetos da classe TField correspondentes aos campos que compõem o índice corrente.

IndexName

Essa propriedade é definida como uma variável do tipo string e define o nome do índice corrente da tabela.

MasterFields

Essa propriedade é definida como uma variável do tipo string e define os nomes dos campos da tabela principal em um relacionamento (separados por um ponto-e-vírgula), devendo ser definida no componente que representa a tabela secundária.

MasterSource

Essa propriedade é definida como um objeto da classe TDataSource e define o nome do componente DataSource ao qual está associado o componente da classe TTable que representa a tabela principal em um relacionamento entre tabelas. Essa propriedade deve ser definida apenas no componente que representa a tabela secundária no relacionamento.

ReadOnly

Essa propriedade é definida como uma variável booleana e define se a tabela pode ser acessada apenas para visualização de registros.

TableName

Essa propriedade é definida como uma variável do tipo string e define os nomes da tabela representada pelo componente.

TableType

Essa propriedade é definida como uma variável do tipo TTableType e pode receber um dos valores descritos a seguir:

- ttDefault: O tipo da tabela será determinado em função da extensão do nome do arquivo.
- TtParadox: A tabela será do tipo Paradox.
- TtDBase: A tabela será do tipo dBASE.
- TtASCII: A tabela será armazenada em um arquivo-texto no formato ASCII.

Se o valor desta for igual a ttDefault, o tipo de tabela será baseado na extensão do nome do arquivo, conforme descrito a seguir.

Extensão	Significado
DB	Paradox
DBF	dBASE
TXT	ASCII

Principais Métodos da Classe TTable

Apresenta-se a seguir uma descrição dos principais métodos da classe TTable, além daqueles herdados das suas classes ancestrais:

AddIndex

Declaração

```
procedure AddIndex(const Name, Fields: string; Options: TIndexOptions);
```

Esse método adiciona um índice à tabela, recebendo como parâmetros:

- Uma string com o nome do índice a ser criado.
- Uma string com os nomes dos campos que formam o novo índice (separados por ponto-e-vírgula).
- Uma variável do tipo TIndexOptions, que representa um conjunto que pode incluir os seguintes elementos:
  - ixPrimary: Se o novo índice for definido como a chave primária da tabela (não se aplica a tabelas no formato dBASE).
  - ixUnique: Se o novo índice não admitir duplicidade de valores.
  - ixDescending: Se os registros forem ordenados alfabeticamente de forma decrescente.
  - ixCaseInsensitive: Se não houver diferenciação entre letras maiúsculas e minúsculas na indexação dos campos (não se aplica a tabelas no formato dBASE).
  - ixExpression: O índice será baseado numa expressão (aplica-se apenas a tabelas no formato dBASE).

ApplyRange

Declaração

```
procedure ApplyRange;
```

Esse método filtra os registros da tabela representada pelo componente, aplicando as condições definidas por chamadas aos métodos SetRangeStart, SetRangeEnd, EditRangeStart e EditRangeEnd.

BatchMove

Declaração

```
function BatchMove(ASource: TBDEDataSet; AMode: TBatchMode): Longint;
```

Esse método move ou copia registros de outra tabela para a tabela representada pelo componente, e recebe como parâmetros:

- Um objeto da classe TBDEDataSet, que define o componente que representa a tabela de onde os registros serão movidos ou copiados.
- Uma variável do tipo TBatchMode, que especifica como essa cópia será feita, e que pode assumir um dos valores apresentados a seguir:
  - batAppend: Adiciona os registros ao final da tabela.
  - batAppendUpdate: Adiciona os registros ao final da tabela e sobrepõe os registros já existentes em que haja duplicidade de valores.
  - batCopy: Copia todos os registros da tabela origem para a tabela representada pelo componente.
  - batDelete: Remove os registros que também existem na tabela de origem.
  - batUpdate: Atualiza os registros da tabela representada pelo componente, com os registros correspondentes existentes na tabela de origem.

CancelRange

Declaração

```
procedure CancelRange;
```

Esse método Remove o filtro aplicado a uma tabela por uma chamada ao seu método ApplyRange.

CreateTable

Declaração

```
procedure CreateTable;
```

Esse método permite a criação de uma tabela em run-time (não confundir com a criação do componente).

Entretanto, antes de se executar uma chamada ao método CreateTable, devem-se definir os valores das seguintes propriedades do componente:

- DataBasename: Define nome do diretório no qual a tabela será armazenada ou o seu alias.
- TableName: Define o nome da tabela a ser criada.
- TableType: Define o tipo da tabela que será criada.
- FieldDefs: Define os campos da tabela. Conforme será visto posteriormente, esse objeto tem alguns métodos importantes, como:
  - Clear: Remove todas as definições de campos da tabela.
  - Add: Adiciona um novo campo à tabela.
- IndexDefs: Objeto da classe TindexDefs que define os índices da tabela. Essa classe tem alguns métodos importantes, como:

- Clear: Remove todas as definições de índices da tabela.
- Add: Adiciona um novo índice à tabela.

DeleteIndex

Declaração

```
procedure DeleteIndex(const Name: string);
```

Esse método remove o índice cujo nome é passado como parâmetro na forma de uma string.

DeleteTable

Declaração

```
procedure DeleteTable;
```

Esse método remove do banco de dados a tabela representada pelo componente, e deve ser empregado com o máximo de cautela.

Antes de se executar esse método, deve-se atribuir o valor False à propriedade Active do componente que representa a tabela (o que também pode ser feito por uma chamada ao seu método Close).

EditRangeEnd

Declaração

```
procedure EditRangeEnd;
```

Esse método efetiva as alterações definidas para um filtro, iniciadas após uma chamada ao método EditRangeStart do componente que representa a tabela.

EditRangeStart

Declaração

```
procedure EditRangeStart;
```

Esse método permite que se alterem as definições para um filtro, alterações estas que serão efetivadas após uma chamada ao método EditRangeEnd do componente que representa a tabela.

EmptyTable

Declaração

```
procedure EmptyTable;
```

Esse método remove todos os registros da tabela representada pelo componente.

FindKey

Declaração

```
function FindKey(const KeyValues: array of const): Boolean;
```

Esse método permite a busca exata de um registro, cujos valores armazenados nos campos que compõem o índice corrente sejam iguais aos valores passados como parâmetros, na forma de um array.

Esse método retorna True se for encontrado um registro que atenda as condições especificadas, e False em caso contrário.

FindNearest

Declaração

```
procedure FindNearest(const KeyValues: array of const);
```

Esse método permite a busca aproximada de um registro, cujos valores armazenados nos campos que compõem o índice corrente sejam os mais aproximados aos valores passados como parâmetros na forma de um array.

GetIndexNames

Declaração

```
procedure GetIndexNames(List: TStrings);
```

Esse método armazena em um objeto da classe TStrings (passado como parâmetro na chamada do método) os nomes dos índices definidos para a tabela representada pelo componente.

GotoCurrent

Declaração

```
procedure GotoCurrent(Table: TTable);
```

Esse método sincroniza o registro corrente dentre os registros manipulados pelo componente que representa a tabela com o registro corrente de outro componente TTable, cujo nome é passado como parâmetro, e que acessa a mesma tabela.

LockTable

Declaração

```
procedure LockTable(LockType: TLockType);
```

Esse método restringe o acesso à tabela por outras aplicações, sendo essa restrição definida como uma variável do tipo TLockType, passada como parâmetro na chamada do método, sendo que uma variável desse tipo pode assumir os seguintes valores:

- taReadLock: A tabela não poderá ser acessada para leitura por outras aplicações.
- taWriteLock: A tabela não poderá ser acessada para escrita por outras aplicações.

RenameTable

Declaração

```
procedure RenameTable(const NewTableName: string);
```

Esse método renomeia a tabela representada pelo componente, e todos os arquivos a ela associados. O novo nome da tabela deve ser passado como parâmetro na forma de uma string.

SetRange

Declaração

```
procedure SetRange(const StartValues, EndValues: array of const);
```

Esse método incorpora a funcionalidade dos métodos SetRangeStart e SetRangeEnd, recebendo como parâmetros dois arrays que indicam, respectivamente, os valores iniciais e finais que definirão o filtro a ser aplicado aos campos que definem o índice corrente da tabela.

SetRangeEnd

Declaração

```
procedure SetRangeEnd;
```

Esse método define o início do trecho de código em que deverão ser definidos os valores finais dos campos para o filtro a ser aplicado aos campos que definem o índice corrente da tabela.

SetRangeStart

Declaração

```
procedure SetRangeStart;
```

Esse método define o início do trecho de código em que deverão ser definidos os valores iniciais dos campos para o filtro a ser aplicado aos campos que definem o índice corrente da tabela.

UnLockTable

Declaração

```
procedure UnlockTable(LockType: TLockType);
```

Esse método remove a restrição aplicada à tabela representada pelo componente, definida previamente por uma chamada ao seu método LockTable:

Esse método recebe como parâmetro uma variável do tipo TLockType, já descrita anteriormente.

A Classe TQuery

A classe TQuery é derivada por herança direta da classe TBDDataSet, sendo normalmente utilizada para se estabelecer uma conexão a uma ou mais tabelas de um banco de dados acessadas usando-se declarações SQL.

Quando o valor da propriedade RequestLive do componente (a ser definida posteriormente) é igual a True, os registros provenientes de uma consulta podem ser editados localmente pelo usuário. Essa característica não deve, no entanto, ser confundida com o recurso de cached updates, definida na classe TBDEDataSet e herdada nessa classe.

Principais Propriedades da Classe TQuery

Apresenta-se a seguir uma descrição das principais propriedades da classe TQuery, além daquelas herdadas das suas classes ancestrais:

Constrained

Essa propriedade é definida como uma variável booleana, e se o seu valor for igual a True não serão permitidas alterações nos registros de tabelas do tipo dBASE e Paradox, que não sejam compatíveis com as condições estabelecidas na cláusula SELECT da declaração SQL que gerou o conjunto de registros.

Essas restrições só se aplicam quando a propriedade RequestLive do componente tem o valor True.

Local

Essa propriedade é definida como uma variável booleana, e define se as tabelas acessadas são tabelas locais (dos tipos Paradox e dBASE).

ParamCount

Essa propriedade é definida como uma variável inteira, e define o número de parâmetros definidos para a Query.

Essa é uma propriedade apenas de leitura, e não pode ter o seu valor diretamente alterado pelo usuário.

Params

Essa propriedade é definida como um array de objetos da classe TParams, que representam individualmente os parâmetros definidos para a Query.

Prepared

Essa propriedade é definida como uma variável booleana, e define se a Query foi preparada para ser executada, de modo a melhorar o seu desempenho.

A preparação de uma Query pode ser feita atribuindo-se o valor True a essa propriedade, ou mediante uma chamada ao seu método Prepare.

RequestLive

Essa propriedade é definida como uma variável booleana, e define se os registros provenientes de uma consulta podem ser editados localmente pelo usuário.

RowsAffected

Essa propriedade é definida como uma variável inteira, e define o número de linhas ou registros atualizados ou removidos pela execução da última declaração SQL.

SQL

Essa propriedade é definida como um objeto da classe TStrings (que é uma lista de strings) na qual deve ser armazenada a declaração SQL a ser executada mediante uma chamada aos métodos Open ou ExecSQL do componente.

Text

Essa propriedade é definida como uma variável do tipo Pchar (string de terminação nula) e define o texto da declaração SQL realmente enviada para ser executada pelo Borland Database Engine.

Unidirecional

Essa propriedade é definida como uma variável booleana, e define se os registros provenientes de uma consulta podem ser navegados ou percorridos em uma única direção.

Principais Métodos da Classe TQuery

Apresenta-se a seguir uma descrição dos principais métodos da classe TQuery, além daqueles herdados das suas classes ancestrais.

ExecSQL

Declaração

```
procedure ExecSQL;
```

Esse método permite a execução de declarações SQL que envolvam a inserção, a remoção e a atualização de registros, isto é, declarações SQL que contêm as cláusulas Insert, Delete e Update.

Declarações SQL que envolvem apenas consultas resultantes da utilização da cláusula SELECT devem ser executadas mediante uma chamada ao método Open do componente.

ParamByName

Declaração

```
function ParamByName(const Value: string): TParam;
```

Esse método permite o acesso individual a parâmetros definidos em uma declaração SQL, sendo o nome de um parâmetro passado na forma de uma string na chamada ao procedimento.

Prepare

Declaração

```
procedure Prepare;
```

Esse método prepara a Query a ser executada, atribuindo o valor True à sua propriedade Prepared.

UnPrepare

Declaração

```
procedure UnPrepare;
```

Esse método cancela a preparação da Query a ser executada, atribuindo o valor False à sua propriedade Prepared.

A Classe TUpdateSQL

A classe TUpdateSQL é derivada por herança direta da classe TDataSetUpdateObject (sendo esta derivada diretamente da classe TComponent), e permite que se definam instruções de inserção (INSERT), deleção (DELETE) e atualização (UPDATE) em registros retornados através de uma consulta SQL, mesmo que esta tenha sido definida como uma consulta apenas de leitura (isto é, a propriedade RequestLive do componente Query é igual a False ou os registros foram transformados em registros apenas de leitura durante a execução do código), desde que a propriedade CachedUpdates do componente Query responsável pela execução da declaração SQL tenha sido definida como True.

A grande vantagem da utilização desse componente está no fato de não haver necessidade de se preocupar com o fato de os registros terem sido gerados apenas para leitura pelo componente Query.

Principais Propriedades da Classe TUpdateSQL

Apresenta-se a seguir uma descrição das principais propriedades da classe TUpdateSQL.

InsertSQL

Essa propriedade é determinada como um objeto da classe TStrings, e permite que se defina o código SQL para inserção de registros em uma tabela.

DeleteSQL

Essa propriedade é determinada como um objeto da classe TStrings, e permite que se defina o código SQL para remoção de registros em uma tabela.

ModifySQL

Essa propriedade é determinada como um objeto da classe TStrings, e permite que se defina o código SQL para a atualização de registros em uma tabela.

Principais Métodos da Classe TUpdateSQL

Apresenta-se a seguir uma descrição dos principais métodos da classe TUpdateSQL.

ExecSQL

Declaração

```
procedure ExecSQL(UpdateKind: TUpdateKind);
```

Esse método executa uma das instruções definidas pelas três propriedades descritas no tópico anterior.

Esse método, ao contrário do método de mesmo nome do objeto Query, recebe como parâmetro uma constante que indica o código a ser executado. A tabela a seguir apresenta essas constantes e seus significados:

- UkModify: Execute a declaração SQL armazenada na propriedade ModifySQL.
- UkInsert: Execute a declaração SQL armazenada na propriedade InsertSQL.
- UkDelete: Execute a declaração SQL armazenada na propriedade DeleteSQL.

Caso as declarações SQL definidas nesse componente tenham parâmetros, os nomes dos parâmetros deverão coincidir com nomes de campos da tabela acessada através do componente Query.

Exemplos de Aplicação

Indexação de Tabelas Acessadas Pelo Componente Table

Existem situações em que precisamos reordenar os registros provenientes de uma tabela, com base nos valores armazenados em um determinado campo (ou conjunto de campos).

Nesses casos, torna-se necessário redefinir o índice corrente da tabela, o que pode ser feito durante a execução da aplicação, conforme será apresentado no exemplo descrito neste tópico.

Chave Primária

Define-se como chave primária um campo (ou conjunto de campos) que tem um valor exclusivo em uma tabela. Toda tabela deve ter ao menos um campo definido como chave primária e, quando isso não for possível, pode-se utilizar mais de um campo, definindo-se o que se chama chave primária composta.

Índices Secundários

Define-se como índice secundário de uma tabela um campo (ou conjunto de campos) pelo qual as informações poderão ser ordenadas. Nesse caso, não existe necessariamente a restrição de valor exclusivo, como ocorre no caso das chaves primárias.

Uma tabela pode ser ordenada pela sua chave primária ou por quaisquer campos que definam uma entidade denominada índice secundário.

Definição do Índice Corrente

No Delphi, a definição do índice corrente de uma tabela acessada usando-se um componente Table pode ser feita da seguinte maneira:

- Na fase de projeto: Definindo-se, diretamente no Object Inspector, o valor da propriedade IndexName do componente Table que representa a tabela (se essa propriedade não for definida, será utilizada a chave primária) ou definindo-se, diretamente no Object Inspector, o valor da propriedade IndexFieldNames do componente Table que representa a tabela.

Conforme já foi descrito anteriormente, as propriedades IndexName e IndexFieldNames são mutuamente excludentes. Atribuir um valor a uma propriedade anula o valor definido para a outra.

- Durante a execução do aplicativo: Definindo-se via código, mediante uma operação de atribuição, o valor das propriedades descritas no tópico anterior. No caso da propriedade IndexFieldNames, os nomes dos campos que formam o índice devem vir separados por ponto-e-vírgula.

Apresenta-se, a seguir, um pequeno exemplo, que utiliza a tabela Employees.db (uma das tabelas-exemplo que acompanham o Delphi 7), na qual a ordenação dos campos é definida durante a execução do aplicativo.

Exemplo de Aplicação

Apresenta-se, a seguir, a descrição das principais propriedades dos componentes utilizados na criação do formulário, o aspecto visual do formulário e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois, em exemplos simples como este, tal fato não gera qualquer tipo de problema.

Nas definições deste exemplo adotou-se uma resolução de 640 x 480.

- Formulário:

  Name: FormIndice
  Width: 500
  Height: 260
  Caption: Exemplo de Definição de Índices em Run-Time
  Position: poScreenCenter

- Table:

  Name: Table1
  DatabaseName: DBDEMOS
  TableName: EMPLOYEE.DB
  Active: True

- DataSource:

  Name: DataSource1
  DataSet: Table1

- DBGrid:

  Name: DBGrid1
  Left: 6
  Top: 80
  Width: 480
  Height: 112
  DataSource: DataSource1

- RadioGroup:

  Name: RadioGroup1
  Left: 56
  Top: 24
  Width: 200
  Height: 40
  Caption: Ordenar por
  Columns: 2
  ItemIndex: 0
  Items. Código,Nome -um em cada linha

- Botão de Comando (BitButton):

  Name: BotaoFechar
  Left: 320
  Top: 32
  Width: 75
  Height: 25
  Kind: bkClose
  Caption: &Fechar

Para definir a propriedade Items do componente RadioGroup, basta digitar os valores desejados na caixa de diálogo String list editor, que é acessada clicando-se com o botão esquerdo do mouse sobre as reticências (...) exibidas à direita do nome da propriedade.

Codificação: A única codificação a ser feita consiste em definir, da seguinte maneira, o procedimento associado ao evento OnClick do componente RadioGroup1:

```
procedure TFormIndice.RadioGroup1Click(Sender: TObject);
begin
     case Radiogroup1.itemindex of
          0 :
            table1.IndexName := '';
          1 :
            table1.IndexName := 'ByName';
     end;
end;
```

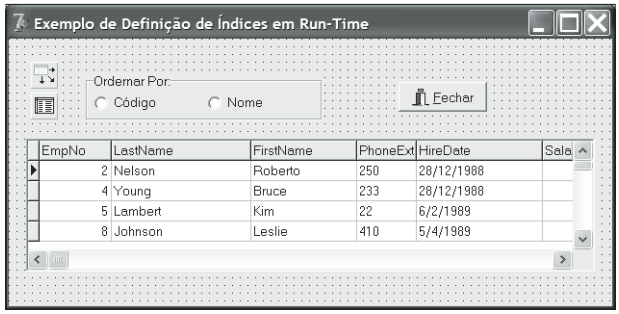


Figura 23.1: Aspecto visual do formulário.

Durante a execução do aplicativo, a seleção de um dos botões de rádio altera a ordem de exibição dos registros.

A estrutura condicional Case verifica o item selecionado e, em função deste, altera a definição do índice corrente.

Apresenta-se a seguir a unit associada ao formulário da aplicação.

```
unit UnitIndice;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ExtCtrls, Grids, DBGrids, Db, DBTables;

type
  TFormIndice = class(TForm)
    Table1: TTable;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    RadioGroup1: TRadioGroup;
    BitBtn1: TBitBtn;
    procedure RadioGroup1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormIndice: TFormIndice;

implementation

{$R *.DFM}

procedure TFormIndice.RadioGroup1Click(Sender: TObject);
begin
     table1.Close;
     case Radiogroup1.itemindex of
          0 :
            table1.IndexName := '';
          1 :
            table1.IndexName := 'ByName';
     end;
     table1.Open;
end;

end.
```

Filtrando os Registros de uma Tabela Acessada Pelo Componente Table

No Delphi, a aplicação de um filtro aos registros de uma tabela acessada usando-se um componente Table pode ser feita da seguinte maneira:

- Na fase de projeto: Definindo-se, diretamente no Object Inspector, o valor da propriedade Filter do componente Table que representa a tabela e definindo-se como True o valor da propriedade Filtered do componente. Para a propriedade Filter, deve-se definir uma expressão booleana, que retorne True ou False.

Ou:

- Durante a execução do aplicativo: Definindo-se via código, mediante uma operação de atribuição, o valor das propriedades descritas anteriormente e codificando-se o procedimento associado ao evento OnFilter Record do componente Table.

Considere que, na tabela Employees.db, usada no exemplo anterior, queremos exibir apenas os empregados contratados após 31/12/92. Nesse caso, devemos digitar, na propriedade Filter do componente Table, a expressão: HireDate > '31/12/92', além de definir como True o valor da sua propriedade Filtered. Simples, não? Considere, no entanto, a seguinte situação: você deseja exibir apenas os dados dos empregados contratados em um determinado mês, e isso será definido durante a execução do aplicativo (usando-se um combobox). Nesse caso, será necessário utilizar a função DecodeDate, para obter o mês de cada registro, e será mais simples digitar o código que define o filtro no procedimento associado ao evento OnFilterRecord do componente Table, além de manter o valor True para a propriedade Filtered.

A fim de tornar este exemplo ainda mais interessante, vamos utilizar um componente checkbox para definir se o filtro deverá ou não ser aplicado à tabela.

Apresenta-se, a seguir, a descrição das principais propriedades dos componentes utilizados na criação do formulário, o aspecto visual do formulário e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois, em exemplos simples como este, tal fato não gera qualquer tipo de problema.

- Formulário:

  Name: FormFiltro:
  Width: 500
  Height: 260
  Caption: Exemplo de Definição de Filtros em Run-Time
  Position: poScreenCenter

- Table:

  Name: Table1
  DatabaseName: DBDEMOS
  TableName: EMPLOYEE.DB
  Active: True

- DataSource:

  Name: DataSource1
  DataSet: Table1

- DBGrid:

  Name: DBGrid1
  Left: 6
  Top: 80
  Width: 480
  Height: 112
  DataSource: DataSource1

- Botão de Comando (BitButton):

  Name: BotaoFechar
  Left: 320
  Top: 32
  Width: 75
  Height: 25
  Kind: bkClose
  Caption: &Fechar

- Combobox:

  Name: ComboBox1
  Left: 60
  Top: 34
  Width: 100
  Height: 21
  Style: csDropDownList
  ItemHeight: 13
  Items: Janeiro, Fevereiro, Março, Abril, Maio, Junho, Julho, Agosto, Setembro, Outubro, Novembro, Dezembro – um em cada linha

- CheckBox1:

  Name: CheckBox1
  Left: 185
  Top: 36
  Width: 90
  Height: 17
  Caption: Aplicar Filtro

Codificação: Como a propriedade ItemIndex do combobox não está disponível no Object Inspector, você deve defini-la codificando da seguinte maneira o procedimento associado ao evento OnCreate do formulário:

```
procedure TFormFiltro.FormCreate(Sender: TObject);
begin
     ComboBox1.ItemIndex := 0;
end;
```

Para alterar a propriedade Filtered do componente Table em função da opção feita pelo usuário, basta codificar da seguinte maneira o procedimento associado ao evento OnClick do CheckBox:

```
procedure TFormFiltro.CheckBox1Click(Sender: TObject);
begin
     Table1.Filtered := CheckBox1.Checked;
end;
```

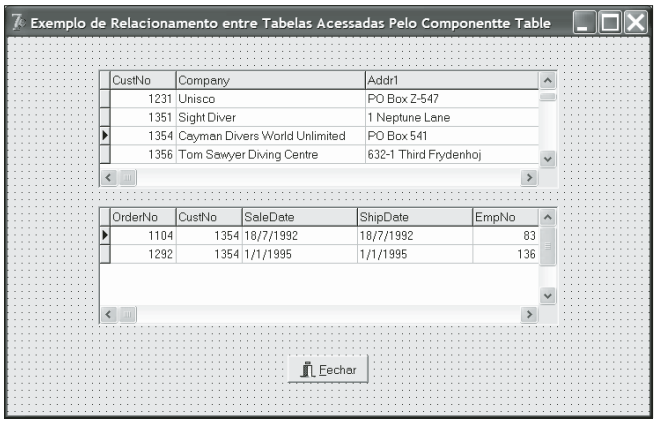


Figura 23.2: Aspecto visual do formulário.

A definição do filtro, nesse caso, deverá ser feita no procedimento associado ao evento OnFilterRecord do componente Table, apresentado a seguir. Nesse procedimento, o parâmetro Accept definirá se o registro deve ou não ser exibido, em função do resultado da expressão booleana definida no seu comando de atribuição.

Neste exemplo, utilizou-se o procedimento DecodeDate, que extrai o dia, mês e ano de um valor do tipo TDateTime. Como a propriedade ItemIndex do combobox começa em 0 e o primeiro mês (Janeiro) começa em 01, basta verificar a diferença entre o valor do mês e o valor armazenado na propriedade ItemIndex do combobox:

```
procedure TFormFiltro.Table1FilterRecord(DataSet : TDataSet;
  var Accept : Boolean);
var
   Dia, Mes, Ano: Word;
begin
    DecodeDate(Table1.FieldByName('HireDate').AsDateTime,Ano,Mes,Dia);
    Accept := Mes - Combobox1.ItemIndex = 1;
end;
```

Para atualizar a exibição dos registros sempre que o usuário alterar a seleção do item no combobox, basta executar o método Refresh do componente Table. Para isso, basta que se codifique, da seguinte maneira, o procedimento associado ao evento OnChange do combobox:

```
procedure TFormFiltro.ComboBox1Change(Sender: TObject);
begin
     Table1.Refresh;
end;
```

Apresenta-se a seguir a codificação completa da unit associada a esse formulário.

```
unit UnitFiltro;

interface
```

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, Grids, DBGrids, Db, DBTables;

type
  TFormFiltro = class(TForm)
    Table1: TTable;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    BitBtn1: TBitBtn;
    ComboBox1: TComboBox;
    CheckBox1: TCheckBox;
    procedure FormCreate(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
    procedure Table1FilterRecord(DataSet: TDataSet; var Accept: Boolean);
    procedure ComboBox1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormFiltro: TFormFiltro;

implementation

{$R *.DFM}

procedure TFormFiltro.FormCreate(Sender: TObject);
begin
     ComboBox1.ItemIndex := 0;
end;

procedure TFormFiltro.CheckBox1Click(Sender: TObject);
begin
     Table1.Filtered := CheckBox1.Checked;
end;

procedure TFormFiltro.Table1FilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
var
   Dia, Mes, Ano: Word;
begin
    DecodeDate(Table1.FieldByName('HireDate').AsDateTime,Ano,Mes,Dia);
    Accept := Mes - Combobox1.ItemIndex = 1;
end;

procedure TFormFiltro.ComboBox1Change(Sender: TObject);
begin
     Table1.Refresh;
end;

end.
```

Campos Calculados

Um campo calculado é um tipo especial de campo de uma tabela, cujo valor depende dos valores armazenados em outros campos. Esse campo só existe durante a execução do aplicativo, reduzindo, dessa maneira, a quantidade de espaço em disco a ser ocupado pela tabela.

Considere novamente a tabela Employees.db. Caso você queira exibir o tempo de cada funcionário na empresa, não há necessidade de criar fisicamente um novo campo na tabela, pois todas as informações necessárias ao cálculo da idade já estão armazenadas na tabela.

Nesse caso, devemos criar um campo calculado, que chamaremos de WorkTime (não sei se esse nome é o mais adequado, mas definirá o tempo do funcionário na empresa). O código que define um campo calculado deve ser definido no procedimento associado ao evento OnCalcFields do componente Table que representa a tabela.

Para criar um campo calculado, você deve executar os seguintes procedimentos:

1. Selecione o componente Table, clicando sobre o mesmo com o botão esquerdo do mouse.
2. Pressione o botão direito do mouse sobre esse componente e selecione o item Fields Editor do menu pop-up, para exibir o editor de campos.
3. Pressione o botão direito do mouse sobre o Fields Editor e selecione o item Add Field no menu pop-up que é exibido. Será exibida a caixa de diálogo Add Fields, mostrada na figura a seguir.

Figura 23.3: A caixa de diálogo Add Fields.

4. Mantendo todos os campos selecionados, clique com o botão esquerdo do mouse sobre o botão OK para fechar essa caixa de diálogo e adicionar os novos campos à janela Editor de Campos (o Fields Editor), como mostra a Figura 23.4.
5. Pressione o botão direito do mouse sobre o Fields Editor e selecione o item New Field no menu pop-up que é exibido. Será exibida a caixa de diálogo New Field, mostrada na Figura 23.5.
6. Selecione o botão de rádio calculated da caixa de diálogo New Field.
7. Digite o nome do novo campo na caixa de texto name da caixa de diálogo New Field (no caso, o nome do campo será WorkTime).
8. Selecione o tipo de campo no combobox Type da caixa de diálogo New Field (no caso, Integer).
9. Digite o tamanho do campo na caixa de texto Size, se for o caso (principalmente em campos alfanuméricos). Para o campo WorkTime, essa caixa de texto deve permanecer em branco. A caixa de diálogo New Fields deve ficar com o aspecto apresentado na figura a seguir.

Figura 23.4: O Fields Editor.

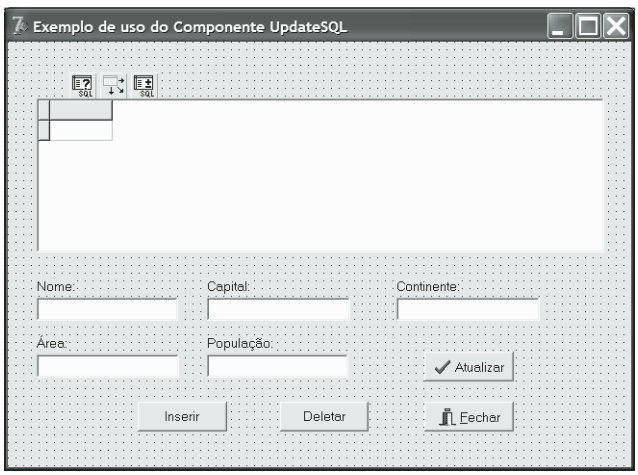


Figura 23.5: A caixa de diálogo New Field.

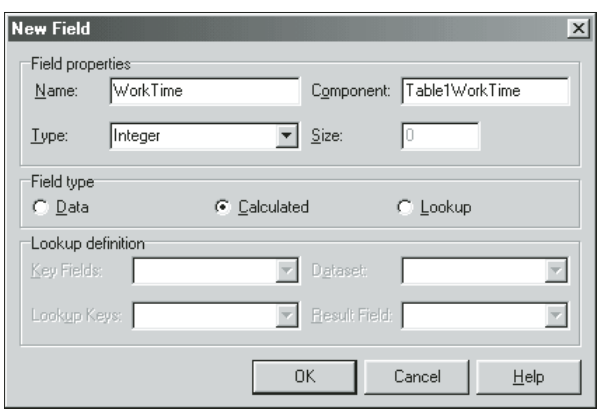


Figura 23.6: A caixa de diálogo New Field, após a digitação das informações necessárias.

10. Selecione o botão OK para fechar a caixa de diálogo New Field e exibir o novo campo no Fields Editor, como mostra a figura a seguir.

Figura 23.7: Criação do campo calculado.

Crie esse campo calculado no exemplo anterior (usado para aplicação de filtros). Para que o campo calculado tenha o seu valor calculado corretamente, basta codificar, da seguinte maneira, o procedimento associado ao evento OnCalcFields do componente Table1:

```
procedure TFormFiltro.Table1CalcFields(DataSet : TDataSet);
begin
     Table1Age.AsInteger = trunc((Date - Table1HireDate.AsDateTime) / 365);
end;
```

Apresenta-se a seguir a codificação completa da unit associada a esse formulário (criado no exemplo anterior), após a inclusão do campo calculado.

```
unit UnitFiltro;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, Grids, DBGrids, Db, DBTables;

type
  TFormFiltro = class(TForm)
    Table1: TTable;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    BitBtn1: TBitBtn;
    ComboBox1: TComboBox;
    CheckBox1: TCheckBox;
    Table1EmpNo: TIntegerField;
    Table1LastName: TStringField;
    Table1FirstName: TStringField;
    Table1PhoneExt: TStringField;
    Table1HireDate: TDateTimeField;
    Table1Salary: TFloatField;
    Table1Age: TIntegerField;
```

```
    procedure FormCreate(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
    procedure Table1FilterRecord(DataSet: TDataSet; var Accept: Boolean);
    procedure ComboBox1Change(Sender: TObject);
    procedure Table1CalcFields(DataSet: TDataSet);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormFiltro: TFormFiltro;

implementation

{$R *.DFM}

procedure TFormFiltro.FormCreate(Sender: TObject);
begin
     ComboBox1.ItemIndex := 0;
end;

procedure TFormFiltro.CheckBox1Click(Sender: TObject);
begin
     Table1.Filtered := CheckBox1.Checked;
end;

procedure TFormFiltro.Table1FilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
var
   Dia, Mes, Ano: Word;
begin
    DecodeDate(Table1.FieldByName('HireDate').AsDateTime,Ano,Mes,Dia);
    Accept := Mes - Combobox1.ItemIndex = 1;
end;

procedure TFormFiltro.ComboBox1Change(Sender: TObject);
begin
     Table1.Refresh;
end;

procedure TFormFiltro.Table1CalcFields(DataSet: TDataSet);
begin
     Table1Age.AsInteger = trunc((Date - Table1HireDate.AsDateTime) / 365);
end;

end.
```

É importante lembrar que o Fields Editor cria, para cada campo, um objeto que o representa. Esse objeto, que pode ser acessado no Object Inspector, recebe um nome composto pelo nome da tabela seguido pelo nome do campo. No caso do campo Age, por exemplo, esse objeto será chamado Table1Age. Para alterar as propriedades desse objeto, basta selecionar seu nome na caixa de seleção de objetos do Object Inspector.

Campos Lookup

Um campo lookup é um tipo especial de campo de uma tabela, cujo valor reflete o valor armazenado em um campo de uma outra tabela, desde que essas duas tabelas tenham um campo com mesmo significado e valores idênticos (ainda que esses campos possam ter nomes distintos). Esse campo só

existe durante a execução do aplicativo, reduzindo, dessa maneira, a quantidade de espaço em disco a ser ocupado pela tabela.

A tabela Orders.db, que acompanha o Delphi, por exemplo, armazena o código de um cliente no campo CustNo (os dados dos clientes estão armazenados na tabela Customers.db). Para exibir o nome do cliente em um DBGrid que acessa a tabela Orders.db, devemos criar um campo lookup nessa tabela, que busque o nome do cliente no campo Company da tabela Customers.db, usando para isso o campo que as tabelas possuem em comum.

Neste exemplo, vamos criar um campo lookup chamado CustName na tabela Orders.db, cujos registros serão exibidos em um DBGrid. Deverão ser utilizados dois componentes Table, sendo que o primeiro acessará a tabela Orders.db e o segundo, a tabela Customer.db.

Apresenta-se, a seguir, a descrição das principais propriedades dos componentes utilizados na criação do formulário e Datamodule, o aspecto visual do formulário e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois, em exemplos simples como este, tal fato não gera qualquer tipo de problema:

- Formulário:

  Name: FormLookup
  Width: 500
  Height: 300
  Caption: Exemplo de Definição de Campos Lookup
  Position: poScreenCenter

- DBGrid1:

  Name: DBGrid1
  Left: 12
  Top: 30
  Width: 468
  Height: 150
  DataSource: Dados.DataSourceOrders

- Botão de Comando:

  Nome: BotaoFechar
  Left: 208
  Top: 220
  Width: 75
  Height: 25
  Kind: bkClose
  Caption: &Fechar

- Datamodule:

  Name: Dados

- Componentes Table: (a serem colocados no Datamodule)

Name: TblOrders
DatabaseName: DBDEMOS
TableName: ORDERS.DB
Active: True
Name: TblCustomers
DatabaseName: DBDEMOS
TableName: CUSTOMER.DB
Active: True

- DataSource:

  Name: DataSourceOrders
  DataSet: TblOrders

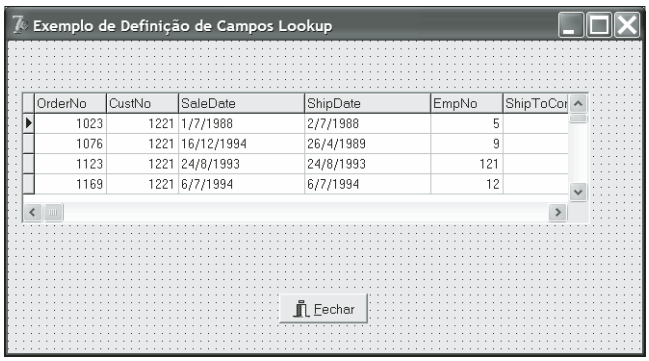


Figura 23.8: Aspecto visual do formulário.

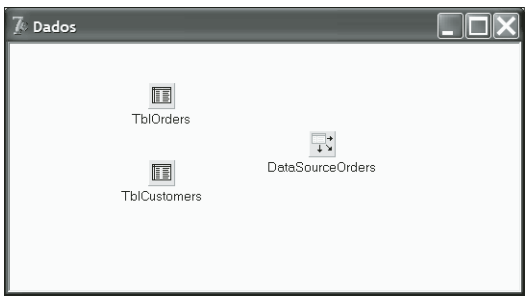


Figura 23.9: Aspecto visual do DataModule.

Para criar um campo lookup, você deve executar os seguintes procedimentos:

1. Selecione o componente TblOrders, clicando sobre o mesmo com o botão esquerdo do mouse.
2. Pressione o botão direito do mouse sobre esse componente e selecione o item Fields Editor do menu pop-up, para exibir o editor de campos.
3. Pressione o botão direito do mouse sobre o Fields Editor e selecione o item Add Field no menu pop-up que é exibido. Será exibida a caixa de diálogo Add Fields.

4. Mantendo todos os campos selecionados, clique com o botão esquerdo do mouse sobre o botão OK para fechar essa caixa de diálogo e adicionar os novos campos ao Fields Editor.
5. Pressione o botão direito do mouse sobre o Fields Editor e selecione o item New Field no menu pop-up que é exibido. Será exibida a caixa de diálogo New Field, mostrada nos tópicos anteriores.
6. Selecione o botão de rádio lookup da caixa de diálogo New Field.
7. Digite o nome do novo campo na caixa de texto name da caixa de diálogo New Field (no caso, o nome do campo será CustName).
8. Selecione o tipo de campo no combobox Type da caixa de diálogo New Field (no caso, String).
9. Digite o tamanho do campo na caixa de texto Size, se for o caso (principalmente em campos alfanuméricos). Para o campo CustName, esse valor deverá ser igual a 30 (que é o tamanho do campo Company na tabela Customers.db).
10. Selecione, no combobox Key Fields, o campo da tabela Orders.db que fará a ligação com a tabela Customer.db (no caso, CustNo).
11. Selecione, no combobox DataSet, o componente que fornecerá o valor do campo que está sendo criado (no caso, TblCustomers).
12. Selecione, no combobox Lookup Keys, o campo da tabela origem (no caso, Customer.db) que fará a ligação com a tabela Orders.db. Nesse caso, o campo desejado é o campo CustNo.
13. Selecione, no combobox Result Field, o campo da tabela origem (no caso, Customer.db) cujo valor será exibido no campo lookup que está sendo criado. Nesse caso, o campo desejado é o campo Company.

A figura a seguir apresenta a caixa de diálogo New Field, após o fornecimento de todas as informações necessárias.

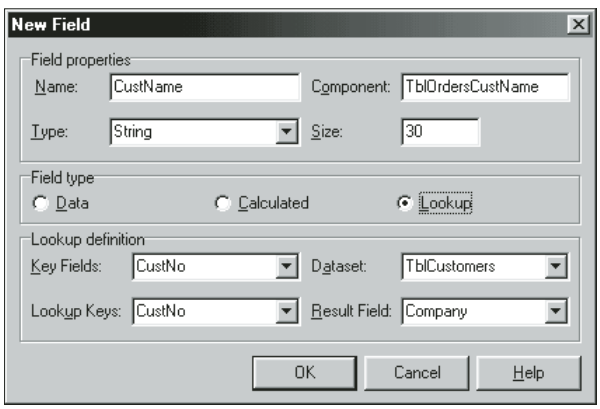


Figura 23.10: A caixa de diálogo New Field, após o fornecimento das informações necessárias à criação de um campo lookup.

14. Selecione o botão OK para fechar a caixa de diálogo New Field. Será criado um objeto chamado Table1Custname, que representará o campo recém-criado.

Para visualizar o campo criado durante a execução da aplicação, use a barra de rolagem horizontal do DBGrid.

Apresentamos a seguir o código completo das units associadas ao formulário e ao Datamodule.

```
unit UnitLookup;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, DBGrids, StdCtrls, Buttons;

type
  TFormLookup = class(TForm)
    DBGrid1: TDBGrid;
    BitBtn1: TBitBtn;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormLookup: TFormLookup;

implementation

uses UnitDadosLookup;

{$R *.DFM}

end.

unit UnitDadosLookup;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Db, DBTables;

type
  TDados = class(TDataModule)
    TblOrders: TTable;
    TblCustomers: TTable;
    DataSourceOrders: TDataSource;
    TblOrdersOrderNo: TFloatField;
    TblOrdersCustNo: TFloatField;
    TblOrdersSaleDate: TDateTimeField;
    TblOrdersShipDate: TDateTimeField;
    TblOrdersEmpNo: TIntegerField;
    TblOrdersShipToContact: TStringField;
    TblOrdersShipToAddr1: TStringField;
    TblOrdersShipToAddr2: TStringField;
    TblOrdersShipToCity: TStringField;
    TblOrdersShipToState: TStringField;
    TblOrdersShipToZip: TStringField;
    TblOrdersShipToCountry: TStringField;
    TblOrdersShipToPhone: TStringField;
```

```
    TblOrdersShipVIA: TStringField;
    TblOrdersPO: TStringField;
    TblOrdersTerms: TStringField;
    TblOrdersPaymentMethod: TStringField;
    TblOrdersItemsTotal: TCurrencyField;
    TblOrdersTaxRate: TFloatField;
    TblOrdersFreight: TCurrencyField;
    TblOrdersAmountPaid: TCurrencyField;
    TblOrdersCustName: TStringField;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Dados: TDados;

implementation

{$R *.DFM}

end.
```

Estabelecendo um Relacionamento Entre Tabelas Representadas Pelo Componente Table

Neste tópico serão apresentados os procedimentos necessários à definição de um relacionamento entre tabelas representadas pelo componente table. Serão utilizadas como exemplo as tabelas Customer.db e Orders.db, que têm um campo em comum – o campo CustNo.

A tabela Customer.db será acessada por um componente Table chamado TblCustomer, enquanto a tabela Orders.db será acessada por outro componente Table chamado TblOrders. Os componentes DataSource associados a esses Tables serão denominados DatasourceCustomer e DatasourceOrders, respectivamente.

No exemplo criado neste tópico, deseja-se exibir, no DBGrid associado à tabela Orders.db, apenas os pedidos do cliente selecionado no DBGrid vinculado à tabela Customers.db. Nesse relacionamento, a tabela Principal (Mestre) será a tabela Customer.db, enquanto a tabela Orders.db será a tabela secundária (ou de detalhes).

Para estabelecer esse relacionamento, você deverá executar os seguintes procedimentos, após inserir os componentes no formulário:

1. Selecione o componente TblOrders e defina a sua propriedade MasterSource como DatasourceCustomer.
2. Defina a propriedade MasterFields desse componente usando a caixa de diálogo Field Links Editor, como mostrado na figura a seguir. Nessa caixa de diálogo, basta selecionar os campos que farão a ligação e o botão Add.

É interessante ressaltar que, nesta nova versão do Delphi, do relacionamento entre tabelas também pode ser estabelecido através da página data diagram do Data Module, conforme será descrito no final deste capítulo.

Apresenta-se, a seguir, a descrição das principais propriedades dos componentes utilizados na criação do formulário e do Datamodule, o aspecto visual do formulário e o código a ser implementado. Alguns

componentes mantiveram seus nomes default, pois, em exemplos simples como este, tal fato não gera qualquer tipo de problema:

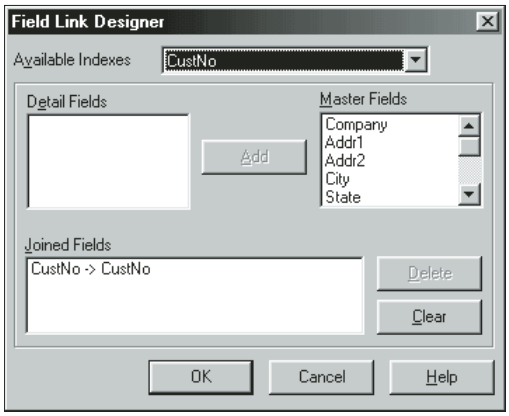


Figura 23.11: A caixa de diálogo Field Link Designer.

- Formulário:

  Name: FormRelacionamento
  Width: 500
  Height: 360
  Caption: Exemplo de Relacionamento entre Tabelas Acessadas Pelo Componentte Table

- Labels:

  Name: Label1
  Left: 12
  Top: 6
  Width: 40
  Height: 13
  Caption: Clientes:
  Name: Label2
  Left: 12
  Top: 136
  Width: 41
  Height: 13
  Caption: Pedidos:

- DBGrids:

  Name: DBGridCustomers
  Left: 12
  Top: 24
  Width: 468
  Height: 110
  DataSource: Dados.DatasourceCustomers
  Name: DBGridOrders
  Left: 12

Top: 150
Width: 468
Height: 110
DataSource: Dados.DatasourceOrders

- Botão de Comando:

  Name: BotaoFechar
  Left: 208
  Top: 288
  Width: 75
  Height: 25
  Caption: &Fechar

- Datamodule:

  Name: Dados

- Componentes Table: (a serem colocados no Datamodule)

  Name: TblOrders
  DatabaseName: DBDEMOS
  TableName: ORDERS.DB
  Active: True
  Name: TblCustomers
  DatabaseName: DBDEMOS
  TableName: CUSTOMER.DB
  Active: True

- DataSource:

  Name: DataSourceOrders
  DataSet: TblOrders
  Name: DatasourceCustomers
  DataSet: TblCustomers

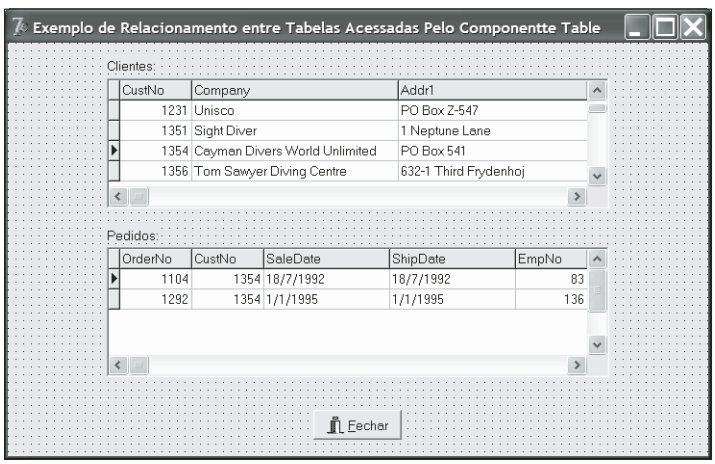


Figura 23.12: Aspecto visual do formulário.

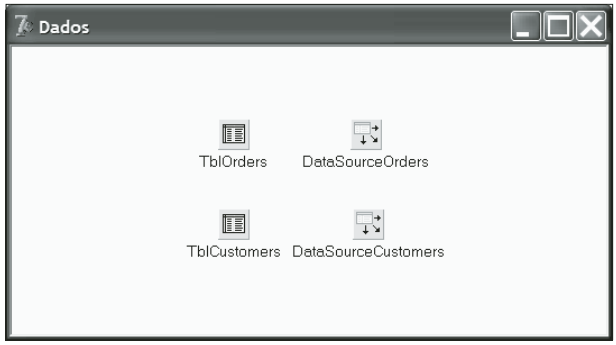


Figura 23.13: Aspecto visual do Datamodule.

Apresenta-se a seguir o código completo das units associadas ao formulário e ao Datamodule. Repare que não é necessária qualquer codificação especial para estabelecer um relacionamento entre tabelas acessadas pelo componente TTable.

```
unit UnitRelacionamento;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, DBGrids, StdCtrls, Buttons;

type
  TFormRelacionamento = class(TForm)
    DBGridCustomers: TDBGrid;
    BitBtn1: TBitBtn;
    DBGridOrders: TDBGrid;
    Label1: TLabel;
    Label2: TLabel;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormRelacionamento: TFormRelacionamento;

implementation

uses UnitDadosRelacionamento;

{$R *.DFM}

end.
var
  FormRelacionamento: TFormRelacionamento;

implementation

{$R *.DFM}

end.

unit UnitDadosRelacionamento;
```

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Db, DBTables;

type
  TDados = class(TDataModule)
    TblOrders: TTable;
    TblCustomers: TTable;
    DataSourceOrders: TDataSource;
    TblOrdersOrderNo: TFloatField;
    TblOrdersCustNo: TFloatField;
    TblOrdersSaleDate: TDateTimeField;
    TblOrdersShipDate: TDateTimeField;
    TblOrdersEmpNo: TIntegerField;
    TblOrdersShipToContact: TStringField;
    TblOrdersShipToAddr1: TStringField;
    TblOrdersShipToAddr2: TStringField;
    TblOrdersShipToCity: TStringField;
    TblOrdersShipToState: TStringField;
    TblOrdersShipToZip: TStringField;
    TblOrdersShipToCountry: TStringField;
    TblOrdersShipToPhone: TStringField;
    TblOrdersShipVIA: TStringField;
    TblOrdersPO: TStringField;
    TblOrdersTerms: TStringField;
    TblOrdersPaymentMethod: TStringField;
    TblOrdersItemsTotal: TCurrencyField;
    TblOrdersTaxRate: TFloatField;
    TblOrdersFreight: TCurrencyField;
    TblOrdersAmountPaid: TCurrencyField;
    TblOrdersCustName: TStringField;
    DataSourceCustomers: TDataSource;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Dados: TDados;

implementation

{$R *.DFM}

end.
```

Pesquisando Registros em Tabelas Representadas Pelo Componente Table

Conforme já descrito anteriormente, o componente Table tem os seguintes métodos de pesquisa:

- FindKey: Esse método permite a busca exata de um registro, cujos valores armazenados nos campos que compõem o índice corrente sejam iguais aos valores passados como parâmetros, na forma de uma array.
- FindNearest: Esse método permite a busca aproximada de um registro, cujos valores armazenados nos campos que compõem o índice corrente sejam os mais aproximados aos valores passados como parâmetros na forma de uma array.

- Locate: Esse método permite a busca exata ou aproximada de um registro, por campos que não façam parte do índice corrente da tabela.

Neste tópico serão utilizados os métodos descritos anteriormente para realizar pesquisas pelos registros da tabela Customer.db.

Apresenta-se, a seguir, a descrição das principais propriedades dos componentes utilizados na criação do formulário e do Datamodule, seus aspectos visuais e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois, em exemplos simples como este, tal fato não gera qualquer tipo de problema:

- Formulário:

  Name: FormPesquisa:
  Width: 500
  Height: 360
  Caption: Exemplo de Pesquisa em Tabelas
  Position: poScreenCenter

- DBGrid:

  Name: DBGridCustomers
  Left: 12
  Top: 24
  Width: 468
  Height: 110
  DataSource: Dados.DatasourceCustomers

- RadioGroups:

  Name: RGTipoPesquisa
  Left: 54
  Top: 200
  Width: 385
  Height: 40
  Caption: Tipo de Pesquisa
  Columns: 3
  ItemIndex: 0
  Items. Exata Indexada, Exata Não-Indexada, Aproximada Indexada
  Name: RGCampo
  Left: 44
  Top: 256
  Width: 385
  Height: 40
  Caption: Campo
  Columns: 3
  ItemIndex: 0
  Items. CustNo, Company, Country

- Caixa de Texto:

  Name: EditPesquisa
  Left: 54
  Top: 264
  Width: 280
  Height: 21

- Botão de Comando:

  Name: BotaoFechar
  Left: 363
  Top: 262
  Width: 75
  Height: 25
  Kind: bkClose
  Caption: &Fechar

- Datamodule:

  Name: Dados

- Table:

  Name: TblCustomer
  DatabaseName: DBDEMOS
  TableName: CUSTOMER.DB
  Active: True

- DataSource:

  Name: DatasourceCustomer
  DataSet: TblCustomer

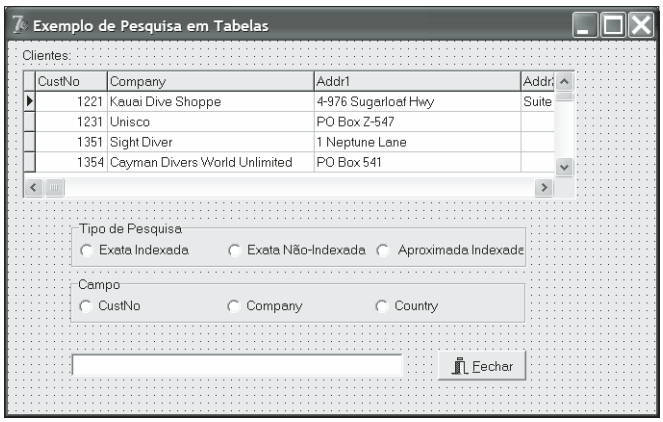


Figura 23.14: Aspecto visual do formulário.

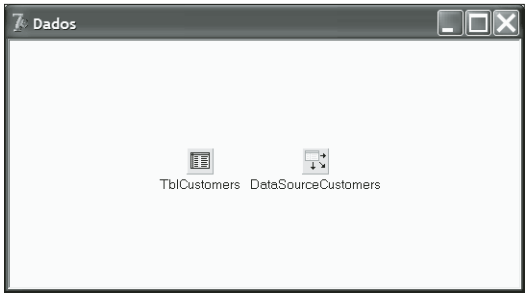


Figura 23.15: Aspecto visual do Datamodule.

Codificação: Neste exemplo, o tipo de pesquisa a ser realizada é definido no procedimento associado ao evento OnClick do componente RGTipoPesquisa. No evento OnClick desse componente, deve-se verificar se uma pesquisa indexada foi selecionada, pois não poderá ser feita uma pesquisa pelo campo Country. Nesse caso, se o campo Country estiver selecionado, este é desmarcado e a seleção passa para o campo CustNo. Para que isso ocorra, deve-se codificar, da seguinte maneira, o procedimento associado ao evento OnClick do componente RGTipoPesquisa:

```
procedure TFormPesquisa.RGTipoPesquisaClick(Sender: TObject);
begin
     case RGTipoPesquisa.ItemIndex of
     0,2 :
       begin
            if RGCampo.ItemIndex = 2 then RGCampo.ItemIndex := 0;
            Dados.TblCustomers.IndexName := '';
       end;
     end;
end;
```

Dependendo do campo selecionado no componente RGCampo, deve-se definir o índice corrente da tabela Customer.db. A única exceção ocorre quando o usuário selecionou uma das opções de pesquisa indexada e escolheu o campo Country, pelo qual a tabela não está indexada. Se esse for o caso, emite-se uma mensagem ao usuário e redefine-se o campo CustNo como o campo de pesquisa.

Para que isso ocorra, deve-se codificar, da seguinte maneira, o procedimento associado ao evento OnClick do componente RGCampo:

```
procedure TFormPesquisa.RGCampoClick(Sender: TObject);
begin
     case RGCampo.ItemIndex of
     0 :
       Dados.TblCustomers.IndexName := '';
     1 :
       Dados.TblCustomers.IndexName := 'ByCompany';
     2 : if RGTipoPesquisa.ItemIndex <> 1 then
        begin
            ShowMessage('Campo Selecionado Não-Indexado');
            RGCampo.ItemIndex := 0;
        end;
     end;
end;
```

Para que a pesquisa seja feita à medida que o usuário digita um valor na caixa de texto, deve-se definir da seguinte maneira o procedimento associado ao evento OnChange do componente EditPesquisa (esse procedimento testa o tipo de pesquisa selecionada e executa o método adequado do componente Table):

```
procedure TFormPesquisa.EditPesquisaChange(Sender: TObject);
begin
     case RGTipoPesquisa.ItemIndex of
          0 :
            Dados.TblCustomers.FindKey([EditPesquisa.Text]);
          1 :
Dados.TblCustomers.Locate(RGCampo.Items[RGCampo.ItemIndex],EditPesquisa.Text,[LoCaseInsensitive]);
          2 :
            Dados.TblCustomers.FindNearest([EditPesquisa.Text]);
     end;
end;
```

Apresenta-se a seguir o código completo da unit associada ao formulário:

```
unit UnitPesquisa;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, DBGrids, StdCtrls, Buttons, ExtCtrls, DB;

type
  TFormPesquisa = class(TForm)
    DBGridCustomers: TDBGrid;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    RGTipoPesquisa: TRadioGroup;
    RGCampo: TRadioGroup;
    EditPesquisa: TEdit;
    procedure RGTipoPesquisaClick(Sender: TObject);
    procedure RGCampoClick(Sender: TObject);
    procedure EditPesquisaChange(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormPesquisa: TFormPesquisa;

implementation

uses UnitDadosRelacionamento;

{$R *.DFM}

procedure TFormPesquisa.RGTipoPesquisaClick(Sender: TObject);
begin
     case RGTipoPesquisa.ItemIndex of
     0,2 :
       begin
            if RGCampo.ItemIndex = 2 then RGCampo.ItemIndex := 0;
            Dados.TblCustomers.IndexName := '';
       end;
     end;
end;

procedure TFormPesquisa.RGCampoClick(Sender: TObject);
begin
     case RGCampo.ItemIndex of
     0 :
```

```
        Dados.TblCustomers.IndexName := '';
      1 :
        Dados.TblCustomers.IndexName := 'ByCompany';
      2 : if RGTipoPesquisa.ItemIndex <> 1 then
         begin
             ShowMessage('Campo Selecionado Não-Indexado');
             RGCampo.ItemIndex := 0;
         end;
      end;
end;

procedure TFormPesquisa.EditPesquisaChange(Sender: TObject);
begin
     case RGTipoPesquisa.ItemIndex of
          0 :
            Dados.TblCustomers.FindKey([EditPesquisa.Text]);
          1 :
     Dados.TblCustomers.Locate(RGCampo.Items[RGCampo.ItemIndex],EditPesquisa.Text,[LoCaseInsensitive]);
          2 :
            Dados.TblCustomers.FindNearest([EditPesquisa.Text]);
     end;
end;

end.
```

Criação de Tabelas em Run-time

Neste tópico será apresentado um exemplo de aplicação que ilustra os procedimentos necessários à criação de tabelas em run-time.

Para criar uma tabela em run-time, deve-se usar o método CreateTable do componente Table.

Entretanto, antes de se executar uma chamada ao método CreateTable do componente Table, os valores das seguintes propriedades devem estar definidos:

- DataBaseName: Define o nome do diretório no qual a tabela será armazenada (ou o seu alias).
- TableName: Define o nome da tabela.
- TableType: Define o tipo da tabela.
- FieldDefs: Essa propriedade é um objeto da classe TFieldDefs, e define os campos da tabela. Essa classe tem alguns métodos importantes, como:
  - Clear, que remove todas as definições de campos da tabela.
  - Add, que adiciona um novo campo à tabela. Esse método recebe como parâmetros: uma string que define o nome do campo, uma constante do tipo TFieldType, que define o tipo do campo. O parâmetro TFieldType pode receber um dos seguintes valores: ftUnknown, ftString, ftSmallint, ftInteger, ftWord, ftBoolean, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftBytes, ftVarBytes, ftAutoInc, ftBlob, ftMemo, ftGraphic, ftFmtMemo, ftParadoxOle, ftDBaseOle, ftTypedBinary, ftCursor, o tamanho do campo (se for o caso) e uma constante booleana, que define se o campo será ou não um campo requerido.

A propriedade Count dessa classe retorna o número de campos definidos no objeto:

- IndexDefs: Essa propriedade é um objeto da classe TIndexDefs e define os índices da tabela. Essa classe tem alguns métodos importantes, como:
  - Clear: Remove todas as definições de índices da tabela.
  - Add: Adiciona um novo índice à tabela. Esse método recebe como parâmetros uma string com o nome do índice; uma string com os nomes dos campos que compõem o índice (separados por ponto-e-vírgula) e um conjunto de opções, em que cada elemento é um dos valores possíveis definidos para o tipo TIndexDefOptions, listados a seguir.

Valor	Significado
ixPrimary	O índice será a chave primária da tabela.
ixUnique	Não pode haver duplicidade de valores.
ixDescending	A indexação será decrescente.
ixExpression	O índice depende de uma expressão-chave do dBASE.
ixCaseInsensitive	O índice não diferencia caracteres maiúsculos e minúsculos (não se aplica ao dBASE).

Apresenta-se, a seguir, a descrição das principais propriedades dos componentes utilizados na criação do formulário, o aspecto visual do formulário e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois em exemplos simples como este tal fato não gera qualquer tipo de problema.

- Formulário:

  Name: FormCriaTabela
  Height: 400
  Width: 435
  Caption: Formulário Para a Criação de Tabelas
  Position: poScreenCenter
  Objetos colocados no formulário

- Table:

  Name: Table1
  DatabaseName: DBDEMOS

- Labels:

  Name: LabelNomeTabela
  Left: 10
  Top: 8
  Width: 179
  Height: 13
  Caption: Digite o Nome da Tabela a Ser Criada

- Caixa de Texto:

  Name: EditNome
  Left: 10

Top: 24
Width: 280
Height: 21

♦ RadioGroups:

Name: RGTipo
Left: 10
Top: 50
Width: 285
Height: 40
Caption: Tipo da Tabela
Columns: 3
ItemIndex: 0
Items. Paradox, DBase, ASCII

♦ GroupBox:

Name: GroupboxDadosDosCampos
Left: 10
Top: 94
Width: 350
Height: 140
Caption: Dados de Cada Campo:
Name: GroupBoxIndices
Left: 10
Top: 240
Width: 350
Height: 100
Caption: Dados dos Índices

♦ Botão de Comando:

Name: BotaoFechar
Left: 317
Top: 58
Width: 75
Height: 25
Kind: bkClose
Caption: &Fechar
Name: BotaoCriaTabela
Left: 138
Top: 344
Width: 150
Height: 25
Caption: Criar Tabela
Kind: bkOK
Objetos colocados dentro do GroupBox GroupboxDadosDosCampos:

- Labels:

  Name: LabelNomeCampo
  Left: 16
  Top: 32
  Width: 84
  Height: 13
  Caption: Nome do campo:
  Name: LabelTipoCampo
  Left: 16
  Top: 68
  Width: 77
  Height: 13
  Caption: Tipo do campo:
  Name: LabelTamanhoCampo
  Left: 16
  Top: 100
  Width: 51
  Height: 13
  Caption: Tamanho:

- Caixa de Texto:

  Name: EditNomeCampo
  Left: 130
  Top: 24
  Width: 120
  Height: 21

- ComboBox:

  Name: ComboBoxTipo
  Left: 145
  Top: 60
  Width: 120
  Height: 21
  Style: csDropDownList
  ItemHeight: 13
  Items. Boolean, Date, DateTime, Float, Integer, SmallInt, String, Time, Word
  Name: EditSize
  Left: 130
  Top: 96
  Width: 40
  Height: 21
  ReadOnly: True
  Text: 0

♦ Objetos UpDown:

Name: UpDown1
Left: 170
Top: 96
Width: 15
Height: 21
Associate: EditSize
Min: 0
Max: 255
Position: 0
Wrap: False

♦ CheckBoxes:

Name: Requerido
Left: 200
Top: 98
Width: 110
Height: 17
Alignment: taLeftJustify
Caption: Campo Requerido

♦ Botão de Comando:

Name: BotaoAddCampo
Left: 290
Top: 45
Width: 120
Height: 25
Caption: Adicionar Campo
Kind: bkOK
Objetos colocados dentro do GroupBoxIndices:

♦ Labels:

Name: LabelCampos
Left: 10
Top: 15
Width: 41
Height: 13
Caption: Campos:
Name: LabelNomeIndice
Left: 145
Top: 15
Width: 75
Height: 13
Caption: Nome do Índice:

- ListBoxes:

  Name: ListBoxCampos
  Left: 10
  Top: 30
  Width: 120
  Height: 65
  ItemHeight: 13
  MultiSelect: True

- CheckBoxes:

  Name: Primaria
  Left: 280
  Top: 32
  Width: 95
  Height: 17
  Caption: Chave Primária
  Name: Decrescente
  Left: 280
  Top: 69
  Width: 95
  Height: 17
  Caption: Decrescente

- Caixa de Texto:

  Name: EditNomeIndice
  Left: 145
  Top: 30
  Width: 120
  Height: 21

- Botão de Comando

  Name: BotaoAddIndice
  Left: 145
  Top: 65
  Width: 150
  Height: 25
  Caption: Adicionar Índice
  Kind: bkOK

Codificação: Para inicializar adequadamente o combobox, defina da seguinte maneira o procedimento associado ao evento OnCreate do formulário:

```
procedure TFormCriaTabelas.FormCreate(Sender: TObject);
begin
     ComboboxTipo.ItemIndex := 0;
     UPDown1.Max := 0;
end;
```

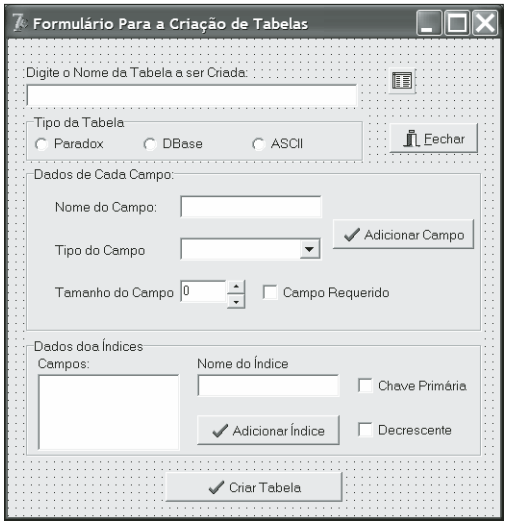


Figura 23.16: Aspecto visual do formulário.

Isso faz com que o combobox exiba o item Boolean quando o formulário é exibido. Como apenas o tipo String pode ter tamanho distinto de 0, as propriedades Text do componente EditSize e as de UpDown1 são inicializadas com o valor 0.

A mudança do item selecionado no combobox é tratada no procedimento associado ao seu evento OnChange, como mostrado a seguir.

```
procedure TFormCriaTabelas.ComboBoxTipoChange(Sender: TObject);
begin
     case ComboboxTipo.ItemIndex of
     6 :
     begin
       EditSize.Text := '10';
       UPDown1.Max := 255;
     end;
     else
         begin
              EditSize.Text := '0';
              UPDown1.Max := 0;
         end;
     end;
end;
```

A criação de campos é tratada no procedimento associado ao evento OnClick do botão BotaoAddCampo, mostrado a seguir.

```
procedure TFormCriaTabelas.BotaoAddCampoClick(Sender: TObject);
var
   Tipo : TFieldType;
begin
     Case comboboxtipo.itemindex of
     0 :
       Tipo := ftBoolean;
     1 :
       Tipo := ftDate;
     2 :
       Tipo := ftDateTime;
     3 :
```

```
        Tipo := ftFloat;
      4 :
        Tipo := ftInteger;
      5 :
        Tipo := ftSmallInt;
      6 :
        Tipo := ftString;
      7 :
        Tipo := ftTime;
      8 :
        Tipo := ftWord;
      end;
Table1.FieldDefs.Add(EditNomeCampo.Text,Tipo,StrToInt(EditSize.Text),Requerido.Checked);
     ListBoxCampos.Items.Add(EditNomeCampo.Text);
     EditNomeCampo.Clear;
end;
```

Inicialmente, o tipo de campo é definido em função do item selecionado no combobox. Depois, o campo é adicionado à tabela usando o método Add da propriedade FieldDefs do componente Table e seu nome é incluído no ListBox que exibe os nomes dos campos. Para finalizar, remove-se o texto exibido no componente EditNomeCampo.

A criação de índices é tratada no procedimento associado ao evento OnClick do botão BotaoAddIndice, mostrado a seguir.

```
procedure TFormCriaTabelas.BotaoAddIndiceClick(Sender: TObject);
var
   i : integer;
   campos : string;
   Opcoes : TIndexOptions;
begin
    campos := '';
    for i :=0 to ListBoxCampos.Items.Count-1 do
        if ListBoxCampos.Selected[i] then
        begin
           campos := campos+ListBoxCampos.Items[i] + ';';
        end;
    campos := copy(campos,1,Length(campos)-1);
    Opcoes := [ixUnique];
    if Primaria.Checked then Opcoes := Opcoes + [ixPrimary];
    if Decrescente.Checked then Opcoes := Opcoes + [ixDescending];
    Table1.IndexDefs.Add(EditNomeIndice.Text,campos,Opcoes);
    EditNomeIndice.Clear;
end;
```

Inicialmente, verificam-se quais os campos selecionados no ListBox e armazenam-se, na variável campos, os nomes dos campos que formam o índice que está sendo criado. Em seguida, define-se o terceiro parâmetro em função da seleção ou não dos CheckBoxes "Primária" e "Descendente".

A criação do índice é feita em uma chamada ao método Add da propriedade IndexDefs do componente Table.

Para finalizar, remove-se o texto exibido no componente EditNomeIndice.

A criação da tabela é tratada no procedimento associado ao evento OnClick do botão BotaoCriaTabela, mostrado a seguir.

```
procedure TFormCriaTabelas.BotaoCriaTabelaClick(Sender: TObject);
begin
```

```
     case RGTipo.ItemIndex of
     0:
        Table1.TableType := ttParadox;
     1 :
        Table1.TableType := ttDBase;
     2 :
        Table1.TableType := ttASCII;
     end;
     Table1.CreateTable;
     ListBoxCampos.Items.Clear;
     EditNome.Clear;
end;
```

Inicialmente, verifica-se qual o tipo selecionado para a tabela. Em seguida, a tabela é criada executando-se o método Createtable do componente Table, e o conteúdo do listbox é removido (bem como o texto do componente EditNome, que define o nome da tabela a ser criada):

Para zerar a lista de campos e índices da tabela que está sendo criada, deve-se executar o método Clear das propriedades FieldDefs e IndexDefs do componente Table, o que é feito no procedimento associado ao evento OnExit do componente EditNome, como mostrado a seguir.

```
procedure TFormCriaTabela.EditNomeExit(Sender: TObject);
begin
     Table1.TableName := EditNome.Text;
     Table1.FieldDefs.Clear;
     Table1.IndexDefs.Clear;
end;
```

Apresenta-se a seguir o código da unit associada a esse formulário:

```
unit UnitCriarTabela;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls, Buttons, ExtCtrls, Db, DBTables;

type
  TFormCriaTabelas = class(TForm)
    Table1: TTable;
    LabelNomeTabela: TLabel;
    EditNome: TEdit;
    RGTipo: TRadioGroup;
    GroupBoxDadosDosCampos: TGroupBox;
    GroupBoxIndices: TGroupBox;
    BitBtn1: TBitBtn;
    BotaoCriaTabela: TBitBtn;
    EditNomeCampo: TEdit;
    ComboBoxTipo: TComboBox;
    EditSize: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    UpDown1: TUpDown;
    Requerido: TCheckBox;
    BotaoAddCampo: TBitBtn;
    ListBoxCampos: TListBox;
    EditNomeIndice: TEdit;
    BotaoAddIndice: TBitBtn;
    LabelCampos: TLabel;
```

```
    LabelNomeIndice: TLabel;
    Primaria: TCheckBox;
    Decrescente: TCheckBox;
    procedure FormCreate(Sender: TObject);
    procedure ComboBoxTipoChange(Sender: TObject);
    procedure BotaoAddCampoClick(Sender: TObject);
    procedure BotaoAddIndiceClick(Sender: TObject);
    procedure BotaoCriaTabelaClick(Sender: TObject);
    procedure EditNomeExit(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormCriaTabelas: TFormCriaTabelas;

implementation


{$R *.DFM}

procedure TFormCriaTabelas.FormCreate(Sender: TObject);
begin
     ComboboxTipo.ItemIndex := 0;
     UPDown1.Max := 0;
end;

procedure TFormCriaTabelas.ComboBoxTipoChange(Sender: TObject);
begin
     case ComboboxTipo.ItemIndex of
     6 :
     begin
       EditSize.Text := '10';
       UPDown1.Max := 255;
     end;
     else
         begin
              EditSize.Text := '0';
              UPDown1.Max := 0;
         end;
     end;
end;

procedure TFormCriaTabelas.BotaoAddCampoClick(Sender: TObject);
var
   Tipo : TFieldType;
begin
     Case comboboxtipo.itemindex of
     0 :
       Tipo := ftBoolean;
     1 :
       Tipo := ftDate;
     2 :
       Tipo := ftDateTime;
     3 :
       Tipo := ftFloat;
     4 :
       Tipo := ftInteger;
     5 :
       Tipo := ftSmallInt;
     6 :
```

```
        Tipo := ftString;
      7 :
        Tipo := ftTime;
      8 :
        Tipo := ftWord;
      end;
Table1.FieldDefs.Add(EditNomeCampo.Text,Tipo,StrToInt(EditSize.Text),Requerido.Checked);
      ListBoxCampos.Items.Add(EditNomeCampo.Text);
      EditNomeCampo.Clear;
end;

procedure TFormCriaTabelas.BotaoAddIndiceClick(Sender: TObject);
var
   i : integer;
   campos : string;
   Opcoes : TIndexOptions;
begin
    campos := '';
    for i :=0 to ListBoxCampos.Items.Count-1 do
        if ListBoxCampos.Selected[i] then
        begin
           campos := campos+ListBoxCampos.Items[i] + ';';
        end;
    campos := copy(campos,1,Length(campos)-1);
    Opcoes := [ixUnique];
    if Primaria.Checked then Opcoes := Opcoes + [ixPrimary];
    if Decrescente.Checked then Opcoes := Opcoes + [ixDescending];
    Table1.IndexDefs.Add(EditNomeIndice.Text,campos,Opcoes);
    EditNomeIndice.Clear;
end;

procedure TFormCriaTabelas.BotaoCriaTabelaClick(Sender: TObject);
begin
     case RGTipo.ItemIndex of
     0:
        Table1.TableType := ttParadox;
     1 :
        Table1.TableType := ttDBase;
     2 :
        Table1.TableType := ttASCII;
     end;
     Table1.CreateTable;
     ListBoxCampos.Items.Clear;
     EditNome.Clear;
end;

procedure TFormCriaTabelas.EditNomeExit(Sender: TObject);
begin
     Table1.TableName := EditNome.Text;
     Table1.FieldDefs.Clear;
     Table1.IndexDefs.Clear;
end;

end.
```

Componentes e Métodos de Navegação

O componente DBNavigator permite a navegação pelos registros de uma tabela, além da inclusão, edição e exclusão de registros.

As principais propriedades desse componente são:

- DataSource: Indica o componente DataSource ao qual estará conectado.
- VisibleButtons: Essa propriedade define os botões a serem exibidos pelo componente, e é muito útil em situações nas quais, por exemplo, desejamos que o componente seja utilizado apenas para fins de navegação pelos registros da tabela.

Você também pode definir os botões que devem ou não ser exibidos por esse tipo de controle, alterando-se o valor da propriedade VisibleButtons durante a execução de um aplicativo. Como essa propriedade é definida por um conjunto, cada elemento representa um dos botões a ser exibido. O valor default dessa propriedade exibe todos os botões, isto é:

```
VisibleButtons:= [nbFirst,nbPrior,nbNext,nbLast,nbInsert,nbDelete,nbEdit,nbPost,nbCancel,nbRefresh]
```

Você pode alterar essa propriedade mediante a inclusão de uma linha de código ou modificando os valores das subpropriedades de VisibleButtons diretamente no Object Inspector.

Para exibir essas subpropriedades, dê um duplo clique com o botão esquerdo do mouse sobre o sinal (+) exibido à esquerda do nome da propriedade VisibleButtons. As subpropriedades são exibidas (podendo ser diretamente alteradas) e o sinal (+) é substituído por (-). Dando um duplo clique com o botão esquerdo do mouse sobre o símbolo (-), as subpropriedades são novamente ocultadas. Para exibir um botão, atribua o valor True à subpropriedade correspondente, e False em caso contrário.

Lembre-se: a existência de um sinal de (+) imediatamente à esquerda do nome de uma propriedade no Object Inspector indica que essa propriedade tem subpropriedades.

Cada subpropriedade (de cima para baixo) corresponde a um dos botões (da esquerda para a direita).

- ShowHint: Essa propriedade é uma variável booleana que define se o componente deve ou não exibir strings de auxílio quando o ponteiro do mouse estiver sobre cada um dos seus botões.
- Hints: Essa propriedade é um objeto da classe TStrings e define as strings de auxílio dos diversos botões exibidos pelo componente DBNavigator (permitindo a sua personalização).

Para personalizar as strings de auxílio do controle DBNavigator, execute os seguintes procedimentos:

1. Selecione o componente DBNavigator, clicando sobre o mesmo com o botão esquerdo do mouse.
2. Selecione a propriedade Hints diretamente no Object Inspector.
3. Clique com o botão esquerdo do mouse sobre as reticências (.) exibidas do lado direito da propriedade no Object Inspector. Será exibida a caixa de diálogo String list editor.
4. Digite as expressões mostradas na Figura 23.14.
5. Selecione o botão OK, para fechar a caixa de diálogo.

Nessa caixa de diálogo, cada string digitada em uma linha corresponde à string de auxílio de um botão. A primeira linha corresponde ao primeiro botão (da esquerda para a direita), a segunda linha ao segundo botão e assim por diante.

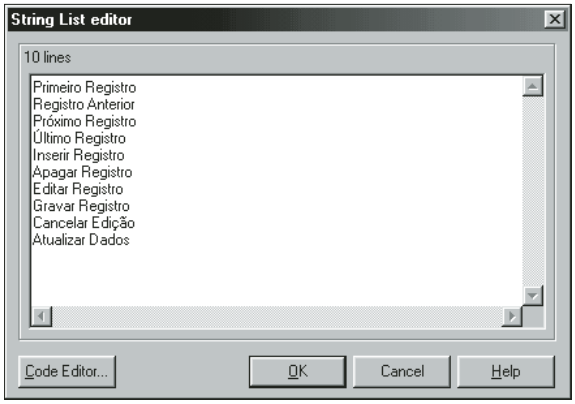


Figura 23.17: Configurando as strings de auxílio para o componente DBNavigator.

Os métodos correspondentes a cada um desses botões, disponíveis para os objetos da classe TTable, são:

- Primeiro Registro: Método First.
- Registro Anterior: Método Prior.
- Próximo Registro: Método Next.
- Último Registro: Método Last.
- Inserir Registro: Método Append.
- Deletar Registro: Método Delete.
- Editar Registro: Método Edit.
- Gravar Registro: Método Post.
- Cancelar Edição do Registro: Método Cancel.
- Atualizar a Exibição do Registro: Método Refresh.

A título de exemplificação, será criado um formulário para acessar e exibir os registros da tabela Customer.db, no qual será inserido um componente DBNavigator e vários botões de comando utilizados como alternativa ao componente DBNavigator.

Apresenta-se, a seguir, a descrição das principais propriedades dos componentes utilizados na criação do formulário, o aspecto visual do formulário e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois em exemplos simples como este tal fato não gera qualquer tipo de problema.

- Formulário:

  Name: FormNavegacao
  Width: 300
  Height: 435
  Caption: Formulário de Navegação
  Position: poScreenCenter

- Table:

  Name: TblCustomers

DatabaseName: DBDEMOS
TableName: CUSTOMER.DB
Active: True

- DataSource:

  Name: DatasourceCustomers
  DataSet: TblCustomers

- DBNavigator:

  Name: DBNavigator1
  Left: 50
  Top: 15
  Width: 240
  Height: 25
  DataSource: DatasourceCustomers

- DBGrid:

  Name: DBGrid1
  Left: 10
  Top: 55
  Width: 410
  Height: 95
  DataSource: DatasourceCustomers

- Botões de Comando:

  Name: BotaoPrimeiro
  Left: 6
  Top: 165
  Width: 100
  Height: 25
  Caption: Primeiro Registro

  Name: BotaoAnterior

  Left: 112
  Top: 165
  Width: 100
  Height: 25
  Caption: Registro Anterior

  Name: BotaoProximo
  Left: 219
  Top: 165
  Width: 100
  Height: 25
  Caption: Próximo Registro

Name: BotaoUltimo
Left: 325
Top: 165
Width: 100
Height: 25
Caption: Último Registro

Name: BotaoAdicionar
Left: 6
Top: 200
Width: 100
Height: 25
Caption: Adicionar Registro

Name: BotaoDeletar
Left: 112
Top: 200
Width: 100
Height: 25
Caption: Deletar Registro

Name: BotaoEditar
Left: 219
Top: 200
Width: 100
Height: 25
Caption: Editar Registro

Name: BotaoGravar
Left: 325
Top: 200
Width: 100
Height: 25
Caption: Gravar Registro

Name: BotaoCancelar
Left: 112
Top: 235
Width: 100
Height: 25
Caption: Cancelar

Name: BotaoAtualizar
Left: 219
Top: 235
Width: 100
Height: 25
Caption: Atualizar

Name: BotaoFechar
Left: 315
Top: 15
Width: 75
Height: 25
Caption: &Fechar
Kind: bkClose

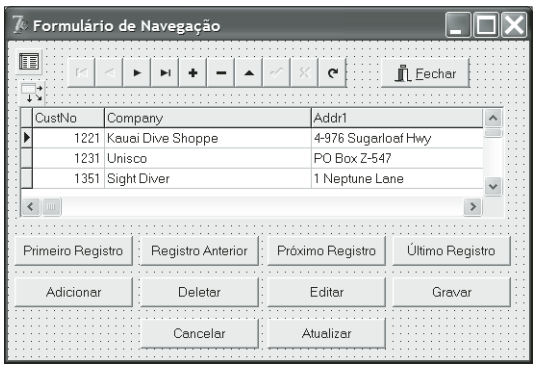


Figura 23.18: Aspecto visual do formulário.

Codificação: O código do procedimento associado ao evento OnClick de cada um dos botões de comando (reproduzidos a seguir) consiste simplesmente em uma chamada a um dos métodos do componente TblCustomer, e dispensa maiores explicações.

```
procedure TFormNavegacao.BotaoPrimeiroClick(Sender: TObject);
begin
    TblCustomers.First;
end;

procedure TFormNavegacao.BotaoAnteriorClick(Sender: TObject);
begin
     TblCustomers.Prior;
end;

procedure TFormNavegacao.BotaoProximoClick(Sender: TObject);
begin
     TblCustomers.Next;
end;

procedure TFormNavegacao.BotaoUltimoClick(Sender: TObject);
begin
     TblCustomers.Last;
end;

procedure TFormNavegacao.BotaoAdicionarClick(Sender: TObject);
begin
     TblCustomers.Append;
end;

procedure TFormNavegacao.BotaoDeletarClick(Sender: TObject);
begin
     TblCustomers.Delete;
end;
```

```
procedure TFormNavegacao.BotaoEditarClick(Sender: TObject);
begin
     TblCustomers.Edit;
end;

procedure TFormNavegacao.BotaoGravarClick(Sender: TObject);
begin
     TblCustomers.Post;
end;

procedure TFormNavegacao.BotaoCancelarClick(Sender: TObject);
begin
     TblCustomers.Cancel;
end;

procedure TFormNavegacao.BotaoAtualizarClick(Sender: TObject);
begin
     TblCustomers.Refresh;
end;
```

Apresenta-se a seguir a codificação completa da unit associada a esse formulário:

```
unit UnitNavegacao;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, Grids, DBGrids, Db, DBTables, ExtCtrls, DBCtrls;

type
  TFormNavegacao = class(TForm)
    DBNavigator1: TDBNavigator;
    TblCustomers: TTable;
    DataSourceCustomers: TDataSource;
    DBGrid1: TDBGrid;
    BotaoPrimeiro: TButton;
    BotaoAnterior: TButton;
    BotaoProximo: TButton;
    BotaoUltimo: TButton;
    BotaoAdicionar: TButton;
    BotaoDeletar: TButton;
    BotaoEditar: TButton;
    BotaoGravar: TButton;
    BotaoCancelar: TButton;
    BotaoAtualizar: TButton;
    BitBtn1: TBitBtn;
    procedure BotaoPrimeiroClick(Sender: TObject);
    procedure BotaoAnteriorClick(Sender: TObject);
    procedure BotaoProximoClick(Sender: TObject);
    procedure BotaoUltimoClick(Sender: TObject);
    procedure BotaoAdicionarClick(Sender: TObject);
    procedure BotaoDeletarClick(Sender: TObject);
    procedure BotaoEditarClick(Sender: TObject);
    procedure BotaoGravarClick(Sender: TObject);
    procedure BotaoCancelarClick(Sender: TObject);
    procedure BotaoAtualizarClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
  FormNavegacao: TFormNavegacao;

implementation

{$R *.DFM}

procedure TFormNavegacao.BotaoPrimeiroClick(Sender: TObject);
begin
    TblCustomers.First;
end;

procedure TFormNavegacao.BotaoAnteriorClick(Sender: TObject);
begin
    TblCustomers.Prior;
end;

procedure TFormNavegacao.BotaoProximoClick(Sender: TObject);
begin
    TblCustomers.Next;
end;

procedure TFormNavegacao.BotaoUltimoClick(Sender: TObject);
begin
    TblCustomers.Last;
end;

procedure TFormNavegacao.BotaoAdicionarClick(Sender: TObject);
begin
    TblCustomers.Append;
end;

procedure TFormNavegacao.BotaoDeletarClick(Sender: TObject);
begin
    TblCustomers.Delete;
end;

procedure TFormNavegacao.BotaoEditarClick(Sender: TObject);
begin
    TblCustomers.Edit;
end;

procedure TFormNavegacao.BotaoGravarClick(Sender: TObject);
begin
    TblCustomers.Post;
end;

procedure TFormNavegacao.BotaoCancelarClick(Sender: TObject);
begin
    TblCustomers.Cancel;
end;
procedure TFormNavegacao.BotaoAtualizarClick(Sender: TObject);
begin
    TblCustomers.Refresh;
end;

end.
```

Tradução da Mensagem Delete Record do Componente Table

Para traduzir a mensagem “Delete Record?”, exibida sempre que se remove um registro de uma tabela, você deve executar os seguintes procedimentos:

1. Atribua o valor False à propriedade ConfirmDelete do componente Table.

2. Digite o seguinte código do procedimento associado ao evento OnDeleteRecord do componente Table:

```
if MessageDlg('Deseja Realmente Deletar o Registro Selecionado
?',mtconfirmation,mbokcancel,0) = mrok
     then
     Abort;
```

Exemplo de Utilização do Componente TSession

Conforme descrito anteriormente, o componente Tsession permite o gerenciamento de várias conexões independentes de bancos de dados em uma única aplicação. Esse componente é muito útil quando se quer controlar o acesso de arquivos paradox em uma rede ou quando se desenvolve uma aplicação de bancos de dados em múltiplas camadas.

O Delphi cria automaticamente um objeto da classe TSession chamado Session em todas as aplicações que acessam bancos de dados mas que não incluem explicitamente esse componente.

Aplicações que acessam simultaneamente tabelas Paradox situadas em endereços distintos de uma rede podem estabelecer múltiplas sessões, uma para cada endereço. Além disso, a utilização de componentes TSession é muito útil quando se quer estabelecer conexões múltiplas e concorrentes com um mesmo banco de dados (como, por exemplo, quando se quer estabelecer duas consultas SQL simultâneas).

Conforme será mostrado no exemplo a seguir, o componente TSession tem métodos (descritos em tópicos anteriores) que permitem obter informações importantes sobre as conexões a um banco de dados. Dentre esses métodos, podem-se destacar:

- AddAlias, que permite que se crie um alias durante a execução do aplicativo.
- AddStandardAlias, semelhante ao método AddAlias, mas que recebe três strings como parâmetros. A primeira define o alias que está sendo criado, a segunda o path a que o Alias se refere e a terceira pode ser igual a um dos seguintes valores: 'Paradox', 'dBASE' ou 'ASCIIDRV'. Para este último parâmetro, uma string nula equivale a 'Paradox'.
- DeleteAlias, que faz justamente o inverso, isto é, remove um alias do BDE.
- GetAliasNames, que coloca em um objeto da classe TStrings, passado como parâmetro na chamada do procedimento, os nomes de todos os alias configurados pelo BDE.
- GetAliasParams, que recebe como primeiro parâmetro o nome de um Alias (na forma de uma string) e retorna, em um objeto da classe TStrings (que deve ser passado como segundo parâmetro), os parâmetros do alias cujo nome foi fornecido.
- IsAlias, que recebe uma string como parâmetro e determina se já existe um alias com o nome definido por essa string.
- GetTableNames, que retorna os nomes das tabelas associadas a um componente TDatabase passado como parâmetro.

No exemplo descrito a seguir, os métodos supracitados serão utilizados na obtenção de informações do BDE.

Apresenta-se, a seguir, a descrição das principais propriedades dos componentes utilizados na criação do formulário, o aspecto visual do formulário e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois, em exemplos simples como este, tal fato não gera qualquer tipo de problema:

- Formulário:

  Name: FormSession
  Width: 350
  Height: 400
  Caption: Exemplo de Uso do Objeto Session

- Botões de Comando:

  Name: BotaoRemove
  Left: 93
  Top: 126
  Width: 150
  Height: 25
  Caption: Remover Alias
  Kind: bkOk
  Name: BotaoFechar
  Left: 133
  Top: 336
  Width: 75
  Height: 25
  Kind: bkClose
  Caption: &Fechar

- Labels:

  Name: Label1
  Left: 36
  Top: 6
  Width: 89
  Height: 13
  Caption: Selecione um alias
  Name: Label2
  Left: 184
  Top: 6
  Width: 140
  Height: 13
  Caption: Tabelas Definidas do alias Selecionado

- ListBoxes:

  Name: ListBoxAlias
  Left: 36
  Top: 21
  Width: 125
  Height: 97

IntegralHeight: True
ItemHeight: 13
Name: ListBoxTabelas
Left: 184
Top: 48
Width: 150
Height: 97
IntegralHeight: True
ItemHeight: 13

- GroupBox:

Name: GroupBox1
Left: 34
Top: 158
Width: 273
Height: 164
Caption: Criação de Alias do Paradox
Objetos colocados dentro do groupBox:

- Labels:

Name: Label3
Left: 16
Top: 24
Width: 31
Height: 13
Caption: Nome
Name: Label4
Left: 16
Top: 47
Width: 39
Height: 13
Caption: Diretório

- Caixas de Texto:

Name: EditNome
Left: 61
Top: 20
Text:
Width: 170
Height: 21

- DirectoryListBox:

Name: ListaDiretorio
Left: 61
Top: 47

Width: 170
Height: 82
ItemHeight: 16

- Botões de Comando:

Name: BotaoCria
Left: 99
Top: 133
Width: 75
Height: 25
Caption: Criar
Kind: bkOK

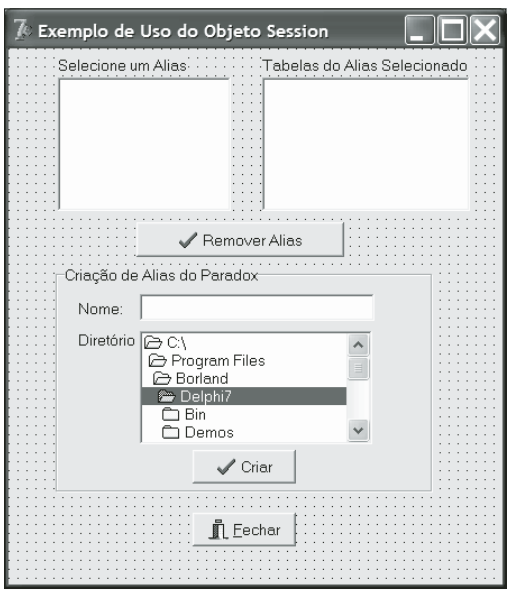


Figura 23.19: Aspecto visual do formulário.

Codificação: A codificação deste exemplo consiste basicamente em chamadas aos métodos do objeto Session, da classe TSession, descritos anteriormente. Nos métodos que necessitam de objetos da classe TStrings como parâmetro, este é passado como a propriedade Items de um ListBox.

São utilizados dois ListBoxes, um para exibir os nomes dos alias e outro para exibir os nomes das tabelas para o alias selecionado.

No procedimento associado ao evento OnCreate do formulário, mostrado a seguir, os ListBoxes são carregados com os nomes dos alias e das tabelas do primeiro alias:

```
procedure TFormSession.FormCreate(Sender: TObject);
begin
     ListaDiretorio.Directory := 'c:\';
     Session.GetAliasNames(ListBoxAlias.Items);
     ListBoxAlias.ItemIndex := 0;
        Session.GetTableNames(ListBoxAlias.Items[ListBoxAlias.ItemIndex],'',false,false,
```

```
        ListBoxTabelas.items);
end;
```

No procedimento associado ao evento OnClick do primeiro ListBox, atualiza-se o conteúdo do segundo, como mostra o trecho de código a seguir.

```
procedure TFormSession.ListBoxAliasClick(Sender: TObject);
begin
     Session.GetTableNames(ListBoxAlias.Items[ListBoxAlias.ItemIndex],'',
false,false,ListBoxTabelas.items);
end;
```

O procedimento associado ao evento OnClick do botão Cria, apresentado a seguir, verifica se o alias a ser criado já existe e, em caso negativo, cria o novo alias com o nome desejado. Por fim, atualiza a lista de alias:

```
procedure TFormSession.BotaocriaClick(Sender: TObject);
begin
     if not(Session.IsAlias(EditNome.Text))
     then Session.AddStandardAlias(EditNome.Text,ListaDiretorio.
Directory,'Paradox')
     else ShowMessage('Já Existe um Alias com esse Nome');
     Session.GetAliasNames(ListBoxAlias.Items);
end;
```

A remoção de um alias é feita no procedimento associado ao evento OnClick do componente BotaoRemove, apresentado a seguir. Inicialmente, o procedimento solicita uma confirmação do usuário e, em caso positivo, elimina o alias selecionado e atualiza a exibição dos ListBoxes.

```
procedure TFormSession.BotaoremoveClick(Sender: TObject);
var
   indice : integer;
begin
     if MessageDlg('Deseja Realmente Deletar o Alias Selecionado
?',mtconfirmation,mbokcancel,0) = mrok
     then
     begin
          indice := ListBoxAlias.ItemIndex-1;
          Session.DeleteAlias(ListBoxAlias.Items[ListBoxAlias.ItemIndex]);
          Session.GetAliasNames(ListBoxAlias.Items);
          ListBoxAlias.ItemIndex := indice;
          Session.GetTableNames(ListBoxAlias.Items[ListBoxAlias.ItemIndex],'',false,false,
          ListBoxTabelas.items);
     end;
end;
```

Para acessar o componente Session, você deve incluir a unit dbTables na cláusula uses da unit do formulário. Repare que, neste exemplo, não incluímos explicitamente os componentes TSession e TDatabase no formulário, e nesses casos a aplicação trata de criar dinamicamente os componentes necessários.

Apresenta-se a seguir o código completo da unit associada ao formulário.

```
unit UnitCriaAlias;

interface

uses
```

```
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  FileCtrl, StdCtrls, Buttons, dbTables;

type
  TFormSession = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    ListBoxAlias: TListBox;
    ListBoxTabelas: TListBox;
    Botaoremove: TBitBtn;
    GroupBox1: TGroupBox;
    Label3: TLabel;
    EditNome: TEdit;
    Label4: TLabel;
    ListaDiretorio: TDirectoryListBox;
    Botaocria: TBitBtn;
    BitBtn3: TBitBtn;
    procedure FormCreate(Sender: TObject);
    procedure ListBoxAliasClick(Sender: TObject);
    procedure BotaocriaClick(Sender: TObject);
    procedure BotaoremoveClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormSession: TFormSession;

implementation

{$R *.DFM}

procedure TFormSession.FormCreate(Sender: TObject);
begin
     ListaDiretorio.Directory := 'c:\';
     Session.GetAliasNames(ListBoxAlias.Items);
     ListBoxAlias.ItemIndex := 0;
        Session.GetTableNames(ListBoxAlias.Items[ListBoxAlias.ItemIndex],'',false,false,
        ListBoxTabelas.items);
end;

procedure TFormSession.ListBoxAliasClick(Sender: TObject);
begin
     Session.GetTableNames(ListBoxAlias.Items[ListBoxAlias.ItemIndex],'',
false,false,ListBoxTabelas.items);
end;

procedure TFormSession.BotaocriaClick(Sender: TObject);
begin
     if not(Session.IsAlias(EditNome.Text))
     then Session.AddStandardAlias(EditNome.Text,ListaDiretorio.
Directory,'Paradox')
     else ShowMessage('Já Existe um Alias com esse Nome');
     Session.GetAliasNames(ListBoxAlias.Items);
end;

procedure TFormSession.BotaoremoveClick(Sender: TObject);
var
   indice : integer;
begin
     if MessageDlg('Deseja Realmente Deletar o Alias Selecionado
?',mtconfirmation,mbokcancel,0) = mrok
```

```
      then
      begin
           indice := ListBoxAlias.ItemIndex-1;
           Session.DeleteAlias(ListBoxAlias.Items[ListBoxAlias.ItemIndex]);
           Session.GetAliasNames(ListBoxAlias.Items);
           ListBoxAlias.ItemIndex := indice;
Session.GetTableNames(ListBoxAlias.Items[ListBoxAlias.ItemIndex],'',false,false,
           ListBoxTabelas.items);
      end;
end;

end.
```

Consulta a Bancos de Dados via Declarações SQL Definidas em Run-Time

O acesso para consulta a bancos de dados via declarações SQL em uma aplicação desenvolvida em Delphi é muito simples. Basta definir corretamente as suas propriedades DataBaseName (que deve ser o nome de um alias ou diretório no qual estão armazenadas as tabelas) e SQL, que é um objeto da classe TStrings que armazena a declaração SQL a ser executada pelo componente.

Existem, no entanto, dois métodos do componente Query que executam tarefas semelhantes em situações ligeiramente distintas – o método Open e o método ExecSQL:

- Open: Esse método abre uma tabela (no caso do componente Table) ou executa uma declaração SQL baseada em uma consulta (no caso do componente Query), tornando-a ativa e fornecendo acesso aos seus registros (e, conforme descrito anteriormente, equivale a definir o valor da sua propriedade Active como True).
- ExecSQL: Esse método deve ser utilizado para executar declarações SQL que alteram o conteúdo das tabelas que são acessadas pelo componente, envolvendo comandos como UPDATE, DELETE e INSERT. O método Open, descrito anteriormente, é usado para executar as declarações SQL que geram consultas, mas não alteram o conteúdo das tabelas (como, por exemplo, as declarações SQL que usam o comando SELECT).

A declaração SQL pode ser definida de duas maneiras:

- Na fase de projeto do aplicativo, digitando-se a declaração SQL na caixa de diálogo String List Editor, exibida quando você seleciona, com o botão esquerdo do mouse, as reticências exibidas à direita do nome da propriedade.
- Durante a execução do aplicativo, usando os métodos da propriedade SQL, que é na realidade um objeto da classe TStrings.

Neste tópico será apresentado um exemplo em que a declaração SQL é definida durante a execução do aplicativo, digitando-se a mesma em um componente Memo.

Apresenta-se, a seguir, a descrição das principais propriedades dos componentes utilizados na criação do formulário, o aspecto visual do formulário e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois, em exemplos simples como este, tal fato não gera qualquer tipo de problema:

- Formulário:

  Name: FormSQL

Width: 435
Height: 300
Caption: Exemplo de Uso de SQL
Position: poScreenCenter

- Labels:

Name: Label1
Left: 24
Top: 136
Width: 118
Height: 13
Caption: Digite a Declaração SQL

- Query:

Name: Query1
DatabaseName: DBDEMOS
Active: False

- DataSource:

Name: DataSource1
DataSet: Query1

- DBGrid:

Name: DBGrid1
Left: 24
Top: 8
Width: 350
Height: 125
DataSource: DataSource1

- Botões de Comando:

Name: BotaoOk
Left: 300
Top: 170
Width: 75
Height: 25
Kind: bkOK
Name: BotaoFechar
Left: 300
Top: 215
Width: 75
Height: 25
Kind: bkClose

♦ Memo:

Name: Memo1
Left: 24
Top: 150
Width: 260
Height: 120
Lines.

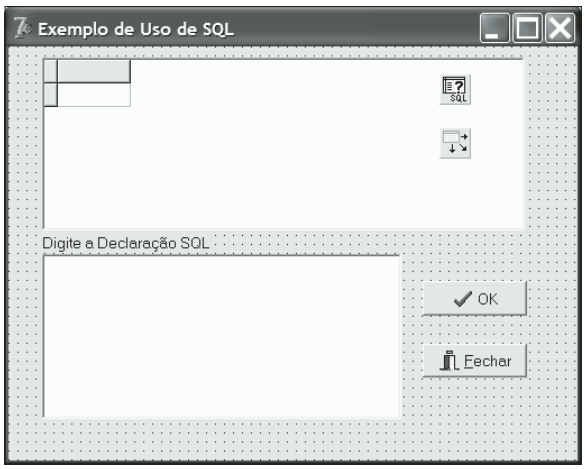


Figura 23.20: Aspecto visual do formulário.

Codificação: A única codificação deste exemplo consiste no procedimento associado ao evento OnClick de um botão de comando, apresentado a seguir.

Nesse procedimento, o texto digitado no Memo (em sua propriedade Lines) é atribuído à propriedade SQL do componente Query1.

Dependendo do fato de o texto representar ou não uma consulta, será executado o método Open ou ExecSQL do componente Query1. A função Pos é usada para testar se a palavra Select está ou não incluída na declaração SQL.

Caso haja erro no código SQL, uma mensagem será exibida para o usuário:

```
procedure TFormSQL.BotaoOkClick(Sender: TObject);
var
   i : integer;
   consulta : boolean;
begin
     Query1.SQL := Memo1.Lines;
     consulta := false;
     for i :=0 to Memo1.Lines.Count-1 do
         if Pos('SELECT',UpperCase(Memo1.Lines[i]))>0 then consulta := true;
     try
        if consulta then Query1.Open else Query1.ExecSQL;
     except
           ShowMessage('Erro no Código SQL');
     end;
end;
```

Repare que, da forma como o exemplo foi preparado, a palvra Select deve ser colocada em uma única linha.

Para corrigir a deficiência deste exemplo, recodifique o procedimento associado ao evento OnClick do botão OK como a seguir:

```
procedure TFormSQL.BotaoOkClick(Sender: TObject);
var
   i : integer;
   declaracao : string;
   consulta : boolean;
begin
     declaracao := '';
     consulta := false;
     for i :=0 to Memo1.Lines.Count-1 do
     if Pos('',Memo1.Lines[i+1]) <> 1
        then declaracao := declaracao + Memo1.Lines[i]
        else declaracao := declaracao + Memo1.Lines[i]+' ';
     Query1.SQL.clear;
     Query1.Sql.Add(declaracao);
     if Pos('SELECT',UpperCase(declaracao))>0 then consulta := true;
     try
        if consulta then Query1.Open else Query1.ExecSQL;
     except
           ShowMessage('Erro no Código SQL');
     end;
end;
```

Deixarei por conta do leitor a compreensão da lógica utilizada.

Nesse caso, pode-se digitar a declaração SQL da maneira mostrada na figura a seguir.

Figura 23.21: Testando a aplicação-exemplo.

Apresenta-se a seguir a codificação completa da unit associada a esse formulário.

```
unit UnitSQL;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, Db, DBTables, Grids, DBGrids;
```

```
type
  TFormSQL = class(TForm)
    DBGrid1: TDBGrid;
    Query1: TQuery;
    DataSource1: TDataSource;
    Memo1: TMemo;
    Label1: TLabel;
    BotaoOk: TBitBtn;
    BotaoFechar: TBitBtn;
    procedure BotaoOkClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormSQL: TFormSQL;

implementation

{$R *.DFM}

procedure TFormSQL.BotaoOkClick(Sender: TObject);
var
   i : integer;
   declaracao : string;
   consulta : boolean;
begin
     declaracao := '';
     consulta := false;
     for i :=0 to Memo1.Lines.Count-1 do
     if Pos('',Memo1.Lines[i+1]) <> 1
        then declaracao := declaracao + Memo1.Lines[i]
        else declaracao := declaracao + Memo1.Lines[i]+' ';
     Query1.SQL.clear;
     Query1.Sql.Add(declaracao);
     if Pos('SELECT',UpperCase(declaracao))>0 then consulta := true;
     try
        if consulta then Query1.Open else Query1.ExecSQL;
     except
           ShowMessage('Erro no Código SQL');
     end;
end;

end.
```

Utilização de Parâmetros em Declarações SQL

Existem situações em que se deseja utilizar um valor variável em uma declaração SQL, também conhecido como parâmetro.

A definição de um parâmetro em uma declaração SQL pode ser feita de duas maneiras:

- Na fase de projeto, acessando a declaração Params do componente Query diretamente no Object Inspector.
- Durante a execução do aplicativo, usando o método AddParams da propriedade Params do componente Query.

Considere, por exemplo, a seguinte declaração SQL:

```
Select * From Country Where Continent =:Nome
```

Essa declaração faz com que só sejam exibidos os registros da tabela Country nos quais o valor do campo Continent seja igual ao parâmetro Nome.

Para definir esse parâmetro na fase de projeto do aplicativo, basta digitar esse código na propriedade SQL do componente Query e selecionar a propriedade Params, para exibir a caixa de diálogo mostrada na figura a seguir, na qual devem ser exibidos os parâmetros da declaração SQL.

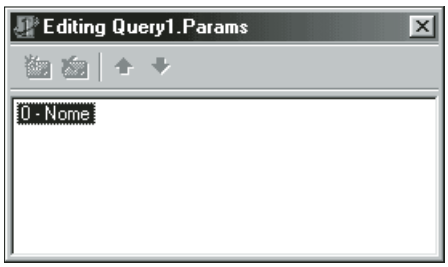


Figura 23.22: A caixa de diálogo para definição de parâmetros.

Uma vez selecionado um parâmetro, suas propriedades podem ser definidas diretamente no Object Inspector. Dentre essas propriedades, destacam-se o tipo do parâmetro e, opcionalmente, um valor default para o mesmo.

Apresenta-se, a seguir, a descrição das principais propriedades dos componentes utilizados na criação do formulário, o aspecto visual do formulário e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois, em exemplos simples como este, tal fato não gera qualquer tipo de problema:

- Formulário:

  Name: FormParametros
  Width: 435
  Height: 220
  Caption: Exemplo do Uso de Parâmetros em Declarações SQL
  Position: poScreenCenter

- Label:

  Name: Label1
  Left: 24
  Top: 140
  Width: 110
  Height: 13
  Caption: Selecione o Continente:

- Componente Query:

  Name: Query1

DatabaseName: DBDEMOS
SQL.Strings: ('Select * From Country Where Continent = Nome')
Active: True

- DataSource:

  Name: DataSource1
  DataSet: Query1

- DBGrid:

  Name: DBGrid1
  Left: 24
  Top: 8
  Width: 350
  Height: 125
  DataSource: DataSource1

- ComboBox:

  Name: ComboBox1
  Left: 25
  Top: 155
  Width: 145
  Height: 21
  Style: csDropDownList
  ItemHeight: 13
  Items: North America, South America

- Botão de Comando:

  Name: BotaoFechar
  Left: 245
  Top: 155
  Width: 75
  Height: 25
  TabOrder: 2
  Kind: bkClose
  Caption: &Fechar

Codificação: A codificação deste exemplo é bastante simples. No procedimento associado ao evento OnCreate do formulário, define-se o valor selecionado no combobox como igual ao valor default do parâmetro:

```
procedure TFormParametros.FormCreate(Sender: TObject);
begin
     ComboBox1.ItemIndex := 1;
end;
```

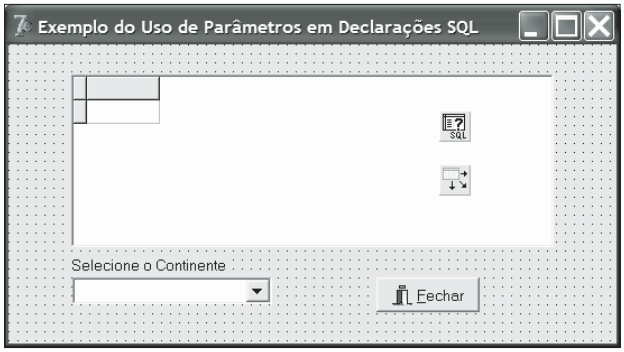


Figura 23.23: Aspecto visual do formulário.

A redefinição do parâmetro e atualização da query é feita no procedimento associado ao evento OnChange do comboBox:

```
procedure TFormParametros.ComboBox1Change(Sender: TObject);
begin
     Query1.Close;
     Query1.ParamByName('Nome').AsString := ComboBox1.Items[ComboBox1.ItemIndex];
     Query1.Prepare;
     Query1.Open;
end;
```

A execução do método Prepare é opcional, mas otimiza a execução do código.

Apresenta-se a seguir a codificação completa da unit associada ao formulário.

```
unit UnitParametroSQL;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, Db, DBTables, Grids, DBGrids;

type
  TFormParametros = class(TForm)
    DBGrid1: TDBGrid;
    Query1: TQuery;
    DataSource1: TDataSource;
    Label1: TLabel;
    BotaoFechar: TBitBtn;
    ComboBox1: TComboBox;
    procedure FormCreate(Sender: TObject);
    procedure ComboBox1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormParametros: TFormParametros;
```

```
implementation

{$R *.DFM}

procedure TFormParametros.FormCreate(Sender: TObject);
begin
     ComboBox1.ItemIndex := 1;
end;

procedure TFormParametros.ComboBox1Change(Sender: TObject);
begin
     Query1.Close;
     Query1.ParamByName('Nome').AsString := ComboBox1.Items[ComboBox1.ItemIndex];
     Query1.Prepare;
     Query1.Open;
end;

end.
```

Utilização do Recurso de Cached Updates

O recurso de cached updates permite que uma aplicação obtenha um conjunto de registros de uma base de dados e faça sua edição local (que pode abranger edição, inclusão ou remoção de registros) e posterior atualização das informações na base de dados.

Além de reduzir o tráfego de informações em redes, esse recurso permite o cancelamento de alterações feitas de forma indesejada.

Para habilitar o recurso de cached updates, basta definir como True o valor da propriedade CachedUpdates do componente Query ou Table.

A atualização dos registros na base de dados é feita mediante uma chamada aos métodos ApplyUpdates e CommitUpdates do componente Query ou Table. O cancelamento das alterações é feito mediante uma chamada ao método CancelUpdates desses mesmos componentes (embora a alteração final só seja garantida após uma chamada ao método Commit do componente TDatabase responsável pela conexão ao banco de dados).

No próximo exemplo, mostraremos como criar uma pequena aplicação que utiliza os recursos de cached updates.

Apresenta-se a seguir a descrição das principais propriedades dos componentes utilizados na criação do formulário, o aspecto visual do formulário e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois, em exemplos simples como este, tal fato não gera qualquer tipo de problema.

- Formulário:

  Name: FormCachedUpdates
  Width: 544
  Height: 375
  Caption: Exemplo de Utilização do Recurso de Cached Updates
  Position: poScreenCenter

- Labels:

  Name: LabelNome
  Left: 8
  Top: 192
  Width: 34
  Height: 13
  Caption: Nome:

  Name: LabelCapital
  Left: 216
  Top: 192
  Width: 41
  Height: 13
  Caption: Capital:

  Name: LabelContinente
  Left: 8
  Top: 224
  Width: 57
  Height: 13
  Caption: Continente:

  Name: LabelArea
  Left: 216
  Top: 224
  Width: 28
  Height: 13
  Caption: Área:

  Name: LabelPopulacao
  Left: 8
  Top: 256
  Width: 57
  Height: 13
  Caption: População:

- Componente Query:

  Name: Query1
  CachedUpdates: True
  DatabaseName: DBDEMOS
  SQL.Strings: ('Select * From Country')
  Active: True
  RequestLive: True

- Componente DataSource:

  Name: DataSource1
  DataSet: Query1

- DBGrid:

  Name: DBGrid1
  Left: 8
  Top: 24
  Width: 460
  Height: 145
  DataSource: DataSource1

- Componentes Caixa de Texto:

  Name: EditNome
  Left: 72
  Top: 188
  Width: 120
  Height: 21

  Name: EditCapital
  Left: 270
  Top: 188
  Width: 120
  Height: 21

  Name: EditContinente
  Left: 72
  Top: 220
  Width: 120
  Height: 21

  Name: EditArea
  Left: 270
  Top: 220
  Width: 120
  Height: 21

  Name: EditPopulacao
  Left: 72
  Top: 252
  Width: 120
  Height: 21

- Botões de Comando:

  Name: BotaoIncluir
  Left: 32
  Top: 296
  Width: 75
  Height: 25
  Caption: Inclui
  TabOrder: 6

Name -= BotaoExcluir
Left: 128
Top: 296
Width: 75
Height: 25
Caption: Excluir

Name: BotaoAplicar
Left: 270
Top: 256
Width: 125
Height: 25
Caption: Aplicar Alterações
Name: BotaoCancela
Left: 288
Top: 296
Width: 125
Height: 25
Caption: Cancelar Alterações

Name: BotaoFechar
Left: 440
Top: 276
Width: 75
Height: 25
Kind: bkClose

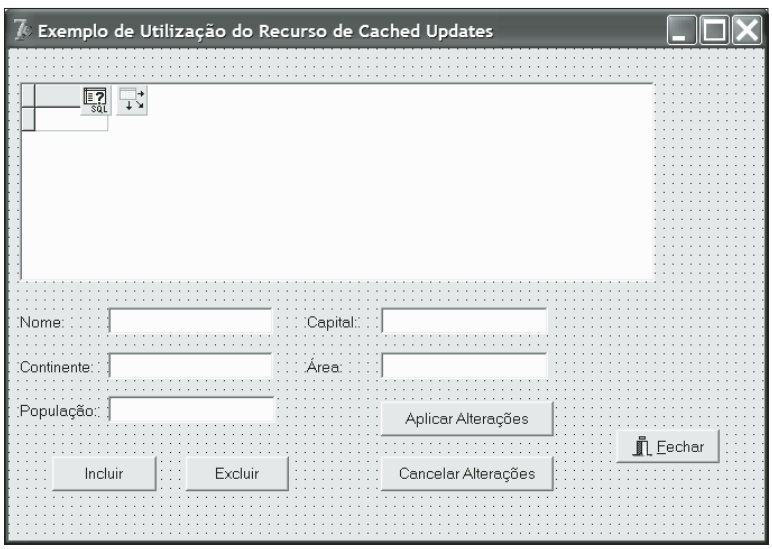


Figura 23.24: Aspecto visual do formulário.

Codificação: Toda a codificação do programa que implementa este exemplo está definida em procedimentos associados ao evento OnClick dos botões de comando. Esses procedimentos são apresentados a seguir e são bastante simples e fáceis de compreender.

```
procedure TFormCachedUpdates.BotaoAplicarClick(Sender: TObject);
begin
    Query1.ApplyUpdates;
    Query1.CommitUpdates;
    Query1.SQL.Clear;
    Query1.SQL.Add('Select * From Country');
    Query1.Open
end;

procedure TFormCachedUpdates.BotaoCancelarClick(Sender: TObject);
begin
    Query1.CancelUpdates;
    Query1.SQL.Clear;
    Query1.SQL.Add('Select * From Country');
    Query1.Open
end;

procedure TFormCachedUpdates.BotaoIncluirClick(Sender: TObject);
begin
    Query1.SQL.Clear;
    Query1.SQL.Add('Insert Into Country(Name,Capital,Area,Population,Continent)');
    Query1.SQL.Add('VALUES(''+EditNome.Text+'',''+EditCapital.Text+'','
+EditArea.Text+'',''+EditPopulacao.Text+'',''+EditContinente.Text+'')');
    Query1.ExecSQL;
    Query1.SQL.Clear;
    Query1.SQL.Add('Select * From Country');
    Query1.Open
end;

procedure TFormCachedUpdates.BotaoExcluirClick(Sender: TObject);
begin
    Query1.Delete;
end;
```

Apresenta-se a seguir o código completo da unit associada ao formulário.

```
unit UnitCache;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
  TFormCachedUpdates = class(TForm)
    LabelNome: TLabel;
    LabelCapital: TLabel;
    LabelContinente: TLabel;
    LabelArea: TLabel;
    LabelPopulacao: TLabel;
    Query1: TQuery;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    EditNome: TEdit;
    EditCapital: TEdit;
    EditContinente: TEdit;
    EditArea: TEdit;
    EditPopulacao: TEdit;
    BotaoFechar: TBitBtn;
    BotaoIncluir: TButton;
    BotaoExcluir: TButton;
    BotaoAplicar: TButton;
    BotaoCancela: TButton;
    procedure BotaoAplicarClick(Sender: TObject);
    procedure BotaoCancelaClick(Sender: TObject);
```

```
    procedure BotaoIncluirClick(Sender: TObject);
    procedure BotaoExcluirClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormCachedUpdates: TFormCachedUpdates;

implementation

{$R *.DFM}

procedure TFormCachedUpdates.BotaoAplicarClick(Sender: TObject);
begin
    Query1.ApplyUpdates;
    Query1.CommitUpdates;
    Query1.SQL.Clear;
    Query1.SQL.Add('Select * From Country');
    Query1.Open
end;

procedure TFormCachedUpdates.BotaoCancelaClick(Sender: TObject);
begin
    Query1.CancelUpdates;
    Query1.SQL.Clear;
    Query1.SQL.Add('Select * From Country');
    Query1.Open
end;

procedure TFormCachedUpdates.BotaoIncluirClick(Sender: TObject);
begin
    Query1.SQL.Clear;
    Query1.SQL.Add('Insert Into Country(Name,Capital,Area,Population,Continent)');
    Query1.SQL.Add('VALUES(''+EditNome.Text+'',''+EditCapital.Text+'','
+EditArea.Text+'',''+EditPopulacao.Text+'',''+EditContinente.Text+'')');
    Query1.ExecSQL;
    Query1.SQL.Clear;
    Query1.SQL.Add('Select * From Country');
    Query1.Open
end;

procedure TFormCachedUpdates.BotaoExcluirClick(Sender: TObject);
begin
    Query1.Delete;
end;

end.
```

EXEMPLO DE UTILIZAÇÃO DO COMPONENTE TUPDATESQL

Conforme já descrito anteriormente, o componente TUpdateSQL permite que se definam instruções de inserção (INSERT), exclusão (DELETE) e atualização (UPDATE) em registros retornados através de uma consulta SQL, mesmo que esta tenha sido definida como uma consulta apenas de leitura (isto é, a propriedade RequestLive do componente Query é igual a False ou os registros foram transformados em registros apenas de leitura durante a execução do código).

Esse componente oferece três propriedades que são objetos da classe TStrings, nas quais devem ser definidos os códigos de inserção, exclusão e atualização. Essas propriedades são denominadas InsertSQL, DeleteSQL e ModifySQL, respectivamente.

Como essas propriedades são objetos da classe TStrings, podem ter o seu valor alterado durante a execução do aplicativo mediante chamadas aos métodos Clear, Add e Insert da classe TStrings.

A execução dessas instruções é realizada por uma chamada ao método ExecSQL do componente UpdateSQL. Esse método, ao contrário do método de mesmo nome do objeto Query, recebe como parâmetro uma constante que indica o código a ser executado. A tabela a seguir apresenta essas constantes e seus significados.

- UkModify: Execute a declaração SQL armazenada na propriedade ModifySQL.
- UkInsert: Execute a declaração SQL armazenada na propriedade InsertSQL.
- UkDelete: Execute a declaração SQL armazenada na propriedade DeleteSQL.

Caso as declarações SQL definidas nesse componente tenham parâmetros, os nomes dos parâmetros deverão coincidir com nomes de campos da tabela acessada através do componente Query.

A grande vantagem da utilização desse componente está no fato de não haver necessidade de se preocupar com o fato de os registros terem sido gerados apenas para leitura pelo componente Query.

Será mostrado a seguir um pequeno exemplo de utilização desse componente.

Apresenta-se inicialmente a descrição das principais propriedades dos componentes utilizados na criação do formulário, o aspecto visual do formulário e o código a ser implementado. Alguns componentes mantiveram seus nomes default, pois, em exemplos simples como este, tal fato não gera qualquer tipo de problema.

- Formulário:

  Name: FormUpdateSQL
  Width: 544
  Height: 375
  Caption: Exemplo de uso do Componente UpdateSQL
  Position: poScreenCenter
  PixelsPerInch: 96

- UpdateSQL:

  Name: UpdateSQL1
  Left: 24
  Top: 16

- Query:

  Name: Query1
  DatabaseName: DBDEMOS
  SQL.Strings: ('Select * From Country')
  UpdateObject: UpdateSQL1
  Active: True

- DataSource:

  Name: DataSource1
  DataSet: Query1

- Labels:

  Name: LabelNome
  Left: 24
  Top: 200
  Width: 34
  Height: 13
  Caption: Nome:

  Name: LabelCapital
  Left: 168
  Top: 200
  Width: 38
  Height: 13
  Caption: Capital:

  Name: LabelContinente
  Left: 328
  Top: 200
  Width: 57
  Height: 13
  Caption: Continente:

  Name: LabelArea
  Left: 24
  Top: 248
  Width: 28
  Height: 13
  Caption: Área:

  Name: LabelPopulacao
  Left: 168
  Top: 248
  Width: 60
  Height: 13
  Caption: População:

- DBGrid:

  Name: DBGrid1
  Left: 24
  Top: 48
  Width: 480
  Height: 130
  DataSource: DataSource1

- Botões de Comando:

  Name: BotaoInsert
  Left: 110
  Top: 304
  Width: 75
  Height: 25
  Caption: Inserir

  Name: BotaoDelete
  Left: 230
  Top: 304
  Width: 75
  Height: 25
  Caption: Deletar

- Caixas de Texto:

  Name: EditNome
  Left: 24
  Top: 216
  Width: 120
  Height: 21

  Name: EditCapital
  Left: 168
  Top: 216
  Width: 120
  Height: 21

  Name: EditContinente
  Left: 328
  Top: 216
  Width: 120
  Height: 21
  Name: EditArea
  Left: 24
  Top: 264
  Width: 120
  Height: 21

  Name: EditPopulacao
  Left: 168
  Top: 264
  Width: 120
  Height: 21

  Name: BotaoAtualiza
  Left: 352
  Top: 262

Width: 75
Height: 25
Kind: bkOk

Name: BotaoFechar
Left: 352
Top: 304
Width: 75
Height: 25
Kind: bkClose
Caption: &Fechar

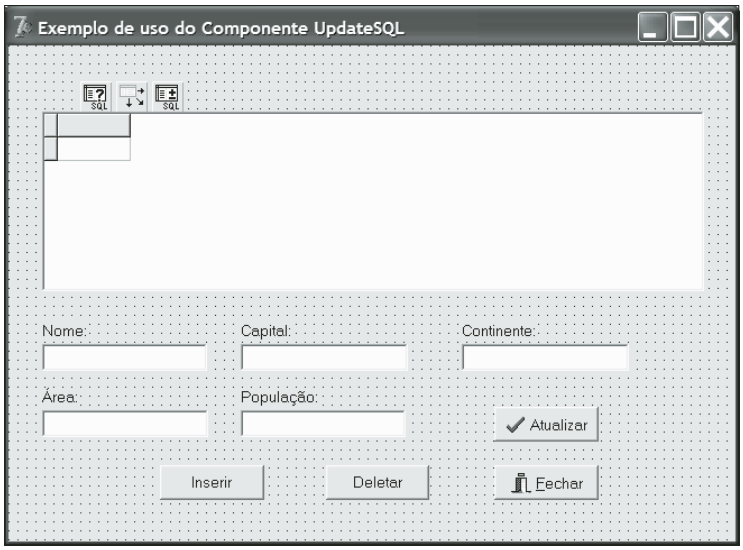


Figura 23.25: Aspecto visual do formulário.

Codificação: Apresenta-se a seguir a codificação deste aplicativo-exemplo, cujo código é muito semelhante aos já apresentados ao longo deste capítulo.

Segue a codificação dos procedimentos associados ao evento OnClick dos botões de comando:

```
procedure T FormUpdateSQL.BotaoInsertClick(Sender: TObject);
begin    UpdateSQL1.InsertSQL.Clear;
     UpdateSQL1.InsertSQL.Add('Insert Into Country(Name,Capital,Area,Population,Continent)');
   UpdateSQL1.InsertSQL.Add('VALUES(''+EditNome.Text+'',''+EditCapital.Text+'','
+EditArea.Text+'',''+EditPopulacao.Text+'','' +EditContinente.Text+'')');
     UpdateSQL1.ExecSQL(ukInsert);
     Query1.disablecontrols;
     Query1.Close;
     Query1.Open;
     Query1.Refresh;
     Query1.enablecontrols;
end;

procedure TFormUpdateSQL.BotaoDeleteClick(Sender: TObject);
begin
     UpdateSQL1.DeleteSQL.Clear;
     UpdateSQL1.DeleteSQL.Add('Delete From Country Where Name =
```

```
"'+Query1.FieldByname('Name').AsString+'"');
     UpdateSQL1.ExecSQL(ukDelete);
     Query1.disablecontrols;
     Query1.Close;
     Query1.Open;
     Query1.Refresh;
     Query1.enablecontrols;
end;
```

Código da unit associada ao formulário:

```
unit UnitUpdateSQL;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, Grids, DBGrids, Db, DBTables;

type
  TFormUpdateSQL = class(TForm)
    LabelNome: TLabel;
    LabelCapital: TLabel;
    LabelContinente: TLabel;
    LabelArea: TLabel;
    LabelPopulacao: TLabel;
    Query1: TQuery;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    EditNome: TEdit;
    EditCapital: TEdit;
    EditContinente: TEdit;
    EditArea: TEdit;
    EditPopulacao: TEdit;
    BotaoFechar: TBitBtn;
    BotaoInsere: TButton;
    BotaoDelete: TButton;
    UpdateSQL1: TUpdateSQL;
    BotaoAtualiza: TBitBtn;
    procedure BotaoAplicarClick(Sender: TObject);
    procedure BotaoCancelaClick(Sender: TObject);
    procedure BotaoInsereClick(Sender: TObject);
    procedure BotaoDeleteClick(Sender: TObject);
    procedure BotaoAtualizaClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormUpdateSQL: TFormUpdateSQL;

implementation

{$R *.DFM}

procedure TFormUpdateSQL.BotaoAplicarClick(Sender: TObject);
begin
    Query1.ApplyUpdates;
    Query1.CommitUpdates;
    Query1.SQL.Clear;
    Query1.SQL.Add('Select * From Country');
```

```
    Query1.Open
end;

procedure TFormUpdateSQL.BotaoCancelaClick(Sender: TObject);
begin
    Query1.CancelUpdates;
    Query1.SQL.Clear;
    Query1.SQL.Add('Select * From Country');
    Query1.Open
end;

procedure TFormUpdateSQL.BotaoInsereClick(Sender: TObject);
begin
     UpdateSQL1.InsertSQL.Clear;
     UpdateSQL1.InsertSQL.Add('Insert Into Country(Name,Capital,Area,Population,Continent)');
   UpdateSQL1.InsertSQL.Add('VALUES(''+EditNome.Text+'',''+EditCapital.Text+'','''
+EditArea.Text+'',''+EditPopulacao.Text+'','' +EditContinente.Text+'')');
     UpdateSQL1.ExecSQL(ukInsert);
     Query1.disablecontrols;
     Query1.Close;
     Query1.Open;
     Query1.Refresh;
     Query1.enablecontrols;
end;

procedure TFormUpdateSQL.BotaoDeleteClick(Sender: TObject);
begin
     UpdateSQL1.DeleteSQL.Clear;
     UpdateSQL1.DeleteSQL.Add('Delete From Country Where Name =
''+Query1.FieldByname('Name').AsString+'''');
     UpdateSQL1.ExecSQL(ukDelete);
     Query1.disablecontrols;
     Query1.Close;
     Query1.Open;
     Query1.Refresh;
     Query1.enablecontrols;
end;

procedure TFormUpdateSQL.BotaoAtualizaClick(Sender: TObject);
begin
     UpdateSQL1.ExecSQL(ukModify);
end;

end.
```

KNOW-HOW EM: APLICAÇÃO DE SENHAS A TABELAS DO TIPO PARADOX

PRÉ-REQUISITOS

- Experiência prévia no emprego do utilitário Database Desktop, para criação e manutenção de tabelas do tipo Paradox.
- Experiência na criação dinâmica de formulários.
- Experiência na utilização de componentes de acesso a banco de dados.

METODOLOGIA

- Apresentação do problema: Aplicação de senhas a tabelas do tipo Paradox.

TÉCNICA

- Apresentação dos procedimentos necessários à aplicação de senhas a tabelas do tipo Paradox em aplicações desenvolvidas com o Borland Delphi.

Definindo Senhas Para uma Tabela do Tipo Paradox

Neste tópico serão apresentados os procedimentos necessários à definição de senhas para tabelas do tipo Paradox, empregando-se o utilitário Database Desktop.

Será utilizada como exemplo a tabela Country.db, uma das tabelas-exemplo que acompanham o Delphi.

Para definir uma senha para a tabela Country.db, você deve executar os seguintes procedimentos:

1. Iniciar a execução do Database Desktop e abrir a tabela Country.db.
2. Selecionar o item Info Restructure, do menu Table, para exibir a caixa de diálogo Restructure Paradox Table.
3. Selecione o item Password Security do combobox Table Properties.
4. Selecione o botão Define. Será exibida a caixa de diálogo Password Security, apresentada na figura a seguir, e na qual deverá ser digitada (e confirmada) uma senha principal para essa tabela. Neste exemplo, usei como senha a palavra AXCEL.

Figura 23.26: A caixa de diálogo Password Security.

5. Selecione o botão Ok para fechar essa caixa de diálogo.

A partir deste momento, quando a tabela for acessada em futuras sessões de utilizações do Database Desktop, será exibida a seguinte caixa de diálogo:

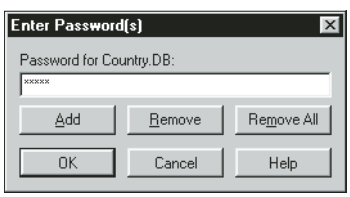


Figura 23.27: A caixa de diálogo Enter Password(s).

Essa caixa de diálogo permite que você digite várias senhas correspondentes a diversas tabelas a serem abertas numa mesma sessão do Database Desktop, bastando digitar cada uma das senhas e o botão Add. Para remover uma ou várias das senhas digitadas, basta selecionar os botões Remove e Remove All.

O único inconveniente é que, nesse caso, sempre que a tabela for aberta a partir de uma aplicação, uma caixa de diálogo semelhante a esta, mostrada na figura a seguir, será exibida. Veremos em um tópico posterior como evitar esse conveniente.

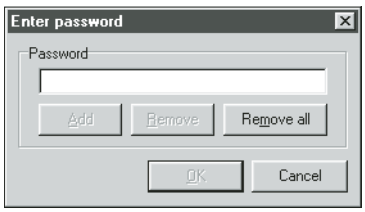


Figura 23.28: A caixa de diálogo Enter Password(s), exibida quando se tenta acessar uma tabela a partir de uma aplicação.

Definindo Senhas Auxiliares Para uma Tabela do Tipo Paradox

No tópico anterior foram apresentados os procedimentos necessários à definição de uma senha principal para uma tabela do tipo Paradox.

Existem situações, no entanto, em que pode ser necessário conceder a outros usuários o acesso a essa tabela, ainda que com privilégios de acesso diferenciados, o que pode ser feito definindo-se senhas auxiliares para a tabela.

Para definir uma senha auxiliar para a tabela Country.db, você deve executar os seguintes procedimentos:

1. Iniciar a execução do utilitário Database Desktop e abrir a tabela Country.db.
2. Selecionar o item Info Restructure, do menu Table, para exibir a caixa de diálogo Restructure Paradox Table.
3. Selecionar o item Password Security do combobox Table Properties.
4. Selecionar o botão Modify. Será exibida a caixa de diálogo Password Security.
5. Selecionar o botão Auxiliary Passwords dessa caixa de diálogo. Será exibida a caixa de diálogo Auxiliary Passwords.

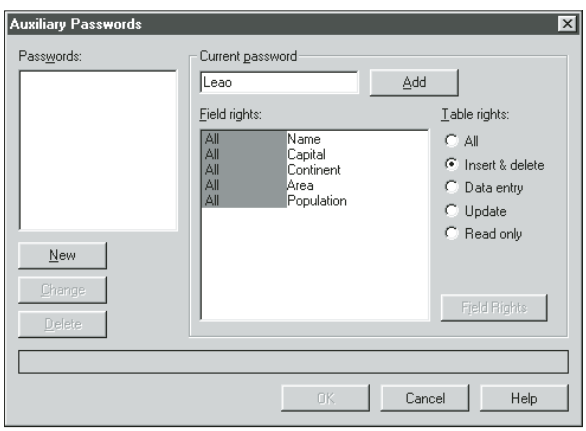


Figura 23.29: A caixa de diálogo Auxiliary Passwords.

Nessa caixa de diálogo você deve digitar cada senha auxiliar, definir os privilégios de acesso (Table Rights) selecionando a opção desejada e o botão Add.

Inibindo a Exibição da Caixa de Diálogo Enter Password Durante a Execução do seu Aplicativo

Conforme descrito anteriormente, existem situações em que se prefere definir uma caixa de diálogo personalizada para obter a senha do usuário em uma caixa de texto.

Para que uma aplicação possa acessar uma tabela sem que seja necessário utilizar a caixa de diálogo Enter Password, basta utilizar o método AddPassword do componente Session, que recebe como parâmetro uma string com um dos possíveis valores para as senhas das tabelas a serem acessadas pela aplicação.

Lembre-se de que, mesmo que não se esteja utilizando explicitamente um componente dessa classe, um objeto dessa classe (denominado Session) será automaticamente criado pelo Delphi.

Protegendo Sua Aplicação Mediante Definição de uma Senha

Para evitar que sua aplicação seja utilizada por usuários não autorizados, você deve executar os seguintes procedimentos:

1. Criar uma tabela chamada Senha.db, com um campo chamado Senha, que pode ser um campo alfanumérico de tamanho igual a 10 caracteres, por exemplo.
2. Definir uma senha para a tabela que armazenará a senha de acesso à aplicação, senha esta que pode ser DELPHI, por exemplo. Isso evita que qualquer pessoa possa abrir a tabela e verificar o valor armazenado no campo.
3. Definir uma senha para a aplicação, e armazenar o seu valor no campo "Senha" da tabela, que só precisará possuir um registro. Neste exemplo definiremos a string AXCEL como sendo esta senha.
4. Inclua a unit DB na cláusula Uses do arquivo de projeto da sua aplicação.
5. No arquivo de projeto da sua aplicação, inclua o seguinte código:

```
Session.AddPassword('AXCEL');
```

Essa linha de código permite que a aplicação tenha acesso irrestrito à tabela.

6. Inclua a unit dbTables na cláusula Uses da unidade de código associada ao formulário em que será feita a verificação da senha.
7. No formulário em que será feita a verificação da senha, declare o seguinte objeto na seção var da unidade de código a ele associada:

```
TabelaSenha: TTable;
```

8. Inclua as seguintes linhas de código no procedimento associado ao evento OnShow do formulário em que será feita a verificação da senha:

```
TabelaSenha := TTable.Create(self);
```

```
TabelaSenha.DatabaseName := 'DBDEMOS';
Tabelasenha.TableName := 'Senha.db';
TabelaSenha.Open;
```

Evidentemente, você deve usar o SEU alias. O alias DBDEMOS foi usado apenas para ilustrar o código necessário.

9. Inclua a seguinte linha de código no procedimento associado ao evento OnHide do formulário em que será feita a verificação da senha:

   ```
   TabelaSenha.Free;
   ```

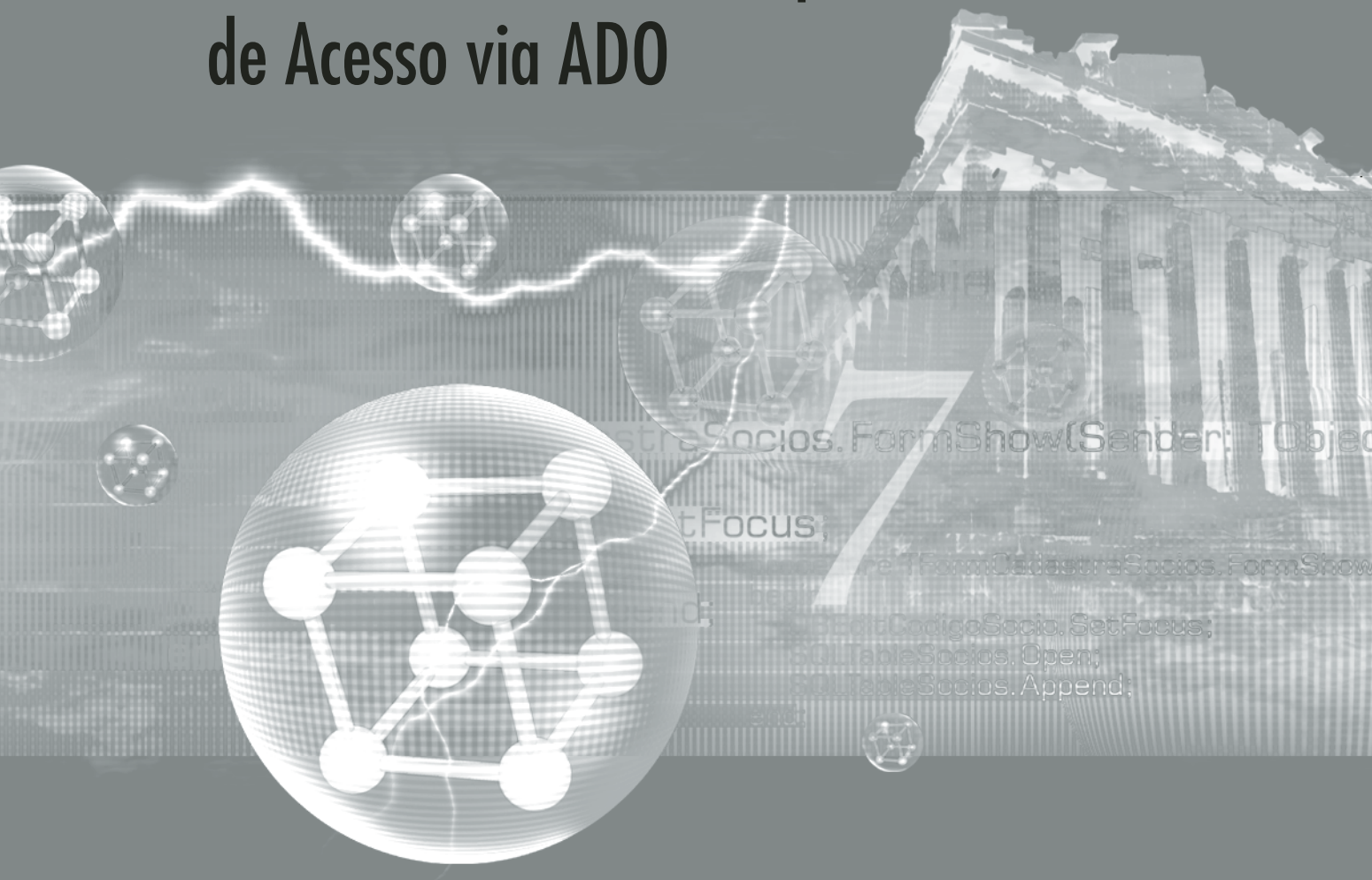
10. No trecho em que o direito de acesso à aplicação deve ser verificado, compare o valor armazenado no campo Senha da tabela Senha.db com o valor fornecido pelo usuário. Considere, por exemplo, que o usuário digite a senha numa caixa de texto chamada "Password".

Essa verificação pode ser feita mediante a inclusão de uma linha de código com a seguinte sintaxe:

```
If TabelaSenha.FieldByName('Senha').AsString = Password.Text
Then
begin
     // Código a ser executado se a senha estiver correta
end
else
begin
     // Código a ser executado se a senha não estiver correta
end;
```

Capítulo 24

Banco de Dados – Componentes de Acesso via ADO



Neste capítulo serão apresentados as classes e os componentes responsáveis pelo acesso a dados via ADO, a partir de uma aplicação desenvolvida com o Borland Delphi 7. Estes componentes não estão disponíveis para aplicações baseadas na CLX, pois a tecnologia ADO só está disponível para o ambiente Windows.

KNOW-HOW em: Classes Fundamentais de Acesso a Bancos de Dados via ADO – A Classe TCustomADODataset e os Componentes TADOConnection, TRDSConnection e TADODataset e TADOCommand

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 7.
- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi 7.
- Compreensão da classe TDataset.

METODOLOGIA

- Apresentação das classes e dos componentes de acesso a bancos de dados via ADO, juntamente com uma descrição das suas propriedades, métodos e eventos.

TÉCNICA

- Descrição das classes e dos componentes de acesso a bancos de dados e apresentação de exemplos de aplicação.

Inicialmente apresentaremos as classes derivadas de TDataset que implementam as principais funcionalidades da tecnologia Activex Data Objects e depois os componentes efetivamente usados no desenvolvimento de aplicações. Estes componentes estão disponíveis desde a versão 5 do Delphi, e por se tratar de uma tecnologia disponível apenas para o ambiente Windows, tais componentes não estão disponíveis para aplicações baseadas na CLX.

O mecanismo definido pela tecnologia Activex Data Objects é bastante complexo, mas a Borland, demonstrando mais uma vez a sua superioridade no que se refere a ferramentas de desenvolvimento, encapsulou este mecanismo em componentes que tornam o seu uso tão fácil como no caso do BDE.

O Componente TADOConnection

Este componente, derivado por herança da classe TCustomConnection, permite que se estabeleça uma conexão a um banco de dados via ADO. Este componente tem uma funcionalidade semelhante ao TDatabase no mecanismo do Borland Database Engine, e permite que diversos componentes de acesso estejam diretamente a ele conectados.

Este componente é o primeiro componente da página ADO da paleta de componentes.

Principais Propriedades da Classe TADOConnection

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TADOConnection, além daquelas herdadas das suas classes-base:

Attributes

Essa propriedade é definida como uma variável do tipo TXactAttributes, e é na verdade um conjunto de duas subproprieades, descritas a seguir:

- xaCommitRetaining: Especificar este valor como True indica que, ao se fazer uma confirmação de uma transação via um comando Commit, uma nova transação é iniciada automaticamente.
- xaAbortRetaining: Especificar este valor como True indica que, ao se fazer um cancelamento de uma transação via um comando Rollback, uma nova transação é iniciada automaticamente.

CommandCount

Essa propriedade é definida como uma variável inteira, e define o número de componentes da classe TADOCommand vinculados a este componente de conexão.

Commands

Essa propriedade é definida como uma array de objetos da classe TCommand, e retorna os objetos da classe TADOCommand vinculados a este componente de conexão que estejam ativos.

CommandTimeOut

Essa propriedade é definida como uma variável inteira, e define o tempo máximo (em segundos) para a execução de um comando (senão considera-se que o comando não foi executado com successo). Seu valor default é igual a 30.

ConnectionObject

Essa propriedade fornece acesso direto ao objeto Activex responsável pela conexão via ADO.

ConnectionString

Essa propriedade é definida como uma variável do tipo WideString e define como será feita a conexão ao banco de dados.

ConnectionTimeOut

Essa propriedade é definida como uma variável inteira, e define o tempo máximo (em segundos) para a ativação de uma conexão (senão considera-se que a conexão não foi executada com sucesso). Seu valor default é igual a 15.

CursorIocation

Essa propriedade é definida como uma variável do tipo TcursorLocation e define se o cursor que aponta para os registros se situa do lado cliente (ideal para manipulação irrestrita dos registros) ou do lado servidor (ideal quando o número de registros a serem acessados é muito grande).

DefaultDatabase

Essa propriedade é definida como uma variável do tipo WideString, e define o banco de dados que a conexão usará como default.

InTransaction

Essa propriedade é definida como uma variável booleana, e indica se uma transação está em progresso.

State

Essa propriedade é definida como uma variável do tipo TobjectSatae, e pode assumir um dos seguintes valores:

- stClosed: Connection: A conexão não está estabelecida e o objeto não está conectado ao banco de dados.
- stOpen: A conexão não está estabelecida mas o objeto está conectado ao banco de dados.
- stConnecting: A conexão está sendo estabelecida.
- stExecuting: Conexão já estabelecida e funcionando.
- stFetching: A conexão está estabelecida mas ainda está recebendo as informações do banco de dados.

Principais Métodos da Classe TADOConnection

Apresenta-se a seguir uma descrição dos principais métodos da classe TADOConnection, além daqueles herdados das suas classes-base.

BeginTrans

Declaração

```
function BeginTrans: Integer;
```

Esse método inicializa uma nova transação com o banco de dados, retornando um valor inteiro indicando o nível da transação. Se a transação for inicializada com sucesso, será executado o procedimento associado ao evento OnBeginTransComplete e a propriedade Instransaction receberá o valor True.

Cancel

Declaração

```
procedure Cancel;
```

Esse método aborta uma tentativa de conexão a um banco de dados.

CloseAllDataSets

Declaração

```
procedure CloseDataSets(All: Boolean = True);
```

Esse método desativa todos os objetos derivados da classe todataset a ele conctados, sem no entanto finalizar a sua própria conexão ao banco de dados.

CommitTrans

Declaração

```
procedure CommitTrans;
```

Esse método realiza um commit sobre a transação corrente.

Execute

Declaração

```
function Execute(const CommandText: WideString; ExecuteOptions: TExecuteOptions = []):
_RecordSet; overload;
procedure Execute(const CommandText: WideString; var RecordsAffected: Integer; ExecuteOptions:
TExecuteOptions = [eoExecuteNoRecords]); overload;
```

Esse método executa um comando sobre o banco de dados. Repare que este método é sobrecarregado, possuindo duas implementações distintas.

O comando a ser executado é passado na forma de uma string, além de outros parâmetros que indicam como os registros devem ser afetados.

GetProcedureNames

Declaração

```
procedure GetProcedureNames(List: TStrings);
```

Esse método armazena na lista de strings passada como parâmetro os nomes de todos os procedimentos armazenados existentes no banco de dados ao qual o componente está conectado.

GetTableNames

Declaração

```
procedure GetTableNames(List: TStrings; SystemTables: Boolean = False);
```

Esse método armazena na lista de strings passada como parâmetro os nomes de todas as tabelas existentes no banco de dados ao qual o componente está conectado.

Open

Declaração

```
procedure Open; overload;
procedure Open(const UserID: WideString; const Password: WideString); overload;
```

Esse método ativa uma conexão ao banco de dados ao qual o componente está conectado. Repare que este método é sobrecarregado, possuindo duas implementações distintas, sendo que a segunda permite que se forneça um nome de usuário e uma senha.

RollbackTrans

Declaração

```
procedure RollbackTrans;
```

Esse método realiza um rollback sobre a transação corrente.

Principais Eventos da Classe TADOConnection

Apresenta-se a seguir uma descrição dos principais eventos da classe TADOConnection, além daqueles herdados das suas classes-base.

OnBeginTransComplete

Este evento ocorre quando uma transação é iniciada.

OnCommitTransComplete

Este evento ocorre quando uma transação é finalizada com um commit.

OnConnectComplete

Este evento ocorre quando uma conexão é estabelecida.

OnDisConnect

Este evento ocorre quando uma conexão é encerrada.

OnExecuteComplete

Este evento ocorre quando um comando é executado com sucesso.

OnInfoMessage

Este evento ocorre quando uma mensagem é enviada do banco de dados para a conexão.

OnRollbackTransComplete

Este evento ocorre quando uma transação é cancelada com um rollback.

OnWillConnect

Este evento ocorre quando uma conexão está para ser estabelecida.

OnWillCommand

Este evento ocorre quando um comando está para ser executado.

O Componente TRDSConnection

Este componente, derivado por herança da classe TCustomConnection, permite que se estabeleça uma conexão a um banco de dados remoto via ADO.

Este componente é o último componente da página ADO da paleta de componentes.

Principais Propriedades da Classe TRDSConnection

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TRDSConnection, além daquelas herdadas das suas classes-base:

Appserver

Essa propriedade é definida como uma variável do tipo OleVariant, e define o nome do servidor de aplicações através do qual a conexão será feita.

ComputerName

Essa propriedade é definida como uma variável do tipo WideString, e define o nome da máquina do servidor de aplicações através do qual a conexão será feita. Pode ser uma URL, sua identidade no IIS ou seu DNS.

InternetTimeOut

Essa propriedade é definida como uma variável inteira, e define o tempo máximo (em milissegundos) para que, se não houver comunicação, a conexão seja desativada (para protocolos http e httpS).

ServerName

Essa propriedade é definida como uma variável do tipo WideString, o ProgID (Identificador) do servidor de aplicações.

Principais Métodos da Classe TRDSConnection

Apresenta-se a seguir uma descrição dos principais métodos da classe TADOConnection, além daquelas herdados das suas classes-base.

GetRecordset

Declaração

```
function GetRecordset(const CommandText: WideString; ConnectionString: WideString = ''): _Recordset;
```

Esse método faz com que o componente receba um conjunto de registros do banco de dados ao qual está conectado, após executar o comando passado como primeiro parâmetro, e usando a string de conexão passada como segundo parâmetro.

Principais Eventos da Classe TRDSConnection

Esta classe não implementa novos eventos, utilizando os herdados das suas classes-base.

A Classe TCustomADODataset

Conforme descrito anteriormente, a classe TDataSet implementa a funcionalidade genérica para acesso a tabelas, sem incorporar as funções da API do Borland Database Engine – o mecanismo de acesso a banco de dados criado pela Borland ou, neste caso, o mecanismo de acesso do Activex Data Objects (ADO).

A classe TCustomADODataSet, derivada por herança direta da classe TDataSet, incorpora as principais características do mecanismo ADO a alguns dos métodos declarados na classe TDataSet, sobrecarregando-os (no caso de métodos virtuais) ou implementando-os (no caso de métodos abstratos).

Os métodos implementados por essa classe não são normalmente usados no desenvolvimento de aplicações em Delphi, pois geralmente são utilizados componentes representados por classes derivadas por herança desta classe.

Principais Propriedades da Classe TCustomADODataSet

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TDataSet.

CacheSize

Essa propriedade é definida como uma variável inteira, e define o número de registros armazenados na memória local pelo componente. Indica portanto o número de registros carregados quando a conexão através do componente se torna ativa.

CanModify

Essa propriedade é definida como uma variável booleana, e define se os registros acessados através do componente podem ou não ser alterados (Na realidade, costuma-se usar a propriedade ReadOnly dos componentes derivados desta classe).

CommandText

Essa propriedade é definida como uma variável do tipo WideString, e define o comando a ser executado (geralmente na forma de uma declaração SQL).

CommandTimeOut

Essa propriedade é definida como uma variável inteira, e define o tempo máximo (em segundos) para a execução de um comando (senão considera-se que o comando não foi executado com sucesso). Seu valor default é igual a 30.

CommandType

Essa propriedade é definida como uma variável do tipo TCommandType, e define o tipo de comando a ser executado.

Esta propriedade pode assumir um dos seguintes valores:

- CmdUnknown: O tipo de comando é desconhecido.

- cmdText: O comando é uma definição textual de uma declaração SQL ou chamada de uma Stored Procedure.
- cmdTable: O comando é, na realidade, o nome de uma tabela.
- cmdStoredProc: O comando é, na realidade, o nome de uma stored procedure.
- cmdFile: O comando se refere ao nome de um arquivo no qual um conjunto de registros está armazenado.
- cmdTableDirect: O comando é, na realidade, o nome de uma tabela, sendo todos os campos retornados.

As constantes cmdTable, cmdTableDirect, e cmdOpenFile não são usadas com o componente TADOCommand.

Connection

Essa propriedade é definida como uma variável do tipo TADOConnection, e define o tipo de coconexão que será usada pelo componente.

ConnectionString

Essa propriedade é definida como uma variável do tipo WideString e define como será feita a conexão ao banco de dados.

As propriedades Connection e ConnectionString são mutuamente excludentes.

IndexFieldCount

Essa propriedade é definida como uma variável inteira, e define o número de campos que definem o índice corrente.

IndexFieldCount

Essa propriedade é definida como uma array de objetos da classe TFields, sendo que cada um destes objetos define um dos campos que formam o índice corrente.

LockType

Essa propriedade é definida como uma variável do tipo TADOLockType e define como os registros são protegidos (travados) ao serem acessados pelo componente. o cursor que aponta para os registros se situa do lado cliente (ideal para manipulação irrestrita dos registros) ou do lado servidor (ideal quando o número de registros a serem acessados é muito grande).

Esta propriedade pode assumir um dos valores descritos a seguir:

- ltUnspecified: Nenhuma proteção especificada.

- ltReadOnly: Os registros só podem ser lidos, sem ser modificados.
- ltPessimistic: Proteção aplicada ao registro corrente.
- ltOptimistic: Proteção aplicada ao registro corrente, mas apenas quando o mesmo está sendo editado.
- ltBatchOptimistic: Usado em operações de transferência de arquivos.

MaxRecords

Essa propriedade é definida como uma variável inteira, e define o número máximo de registros retornados pelo componente.

ParamCheck

Essa propriedade é definida como uma variável booleana, e define se a lista de parâmetros do componente deve ser gerada novamente quando a declaração SQL é alterada durante a execução do aplicativo.

Parameters

Essa propriedade é definida como um objeto da classe TParameters, e define os parâmetros usados em uma declaração SQL.

Prepared

Essa propriedade é definida como uma variável booleana, e define se a declaração SQL deve ser preparada (para otimização da sua execução) artes de ser executada.

RecNo

Essa propriedade é definida como uma variável inteira, e define o número do registro corrente, dentre todos os registros provenientes da tabela representada pelo componente. É uma propriedade apenas de leitura, e não pode ter o seu valor diretamente alterado pelo usuário, podendo ter o seu valor alterado em função da aplicação de um filtro.

RecordCount

Essa propriedade é definida como uma variável inteira, e define o número de registros provenientes da tabela representada pelo componente. É uma propriedade apenas de leitura, e não pode ter o seu valor diretamente alterado pelo usuário, podendo, no entanto, ter o seu valor alterado em função da aplicação de um filtro.

RecordSize

Essa propriedade é definida como uma variável inteira sem sinal (Word), e define o tamanho do registro acessado pelo componente.

RecordStatus

Essa propriedade é definida como uma variável do tipo TRecordStausSet, e define o status do registro corrente (novo, modificado, etc.).

Sort

Essa propriedade é definida como uma variável do tipo Widestring, e define o(s) campo(s) pelos quais os registros acessados pelo componente devem ser ordenados. Os campos devem ser separados por vírgulas, e podem ser acompanhados das palavras ASCENDING ou DESCENDING, indicando se o campo será ordenado de forma ascendente ou descendente. Se não existir um índice para os campos especificados, será criado um índice temporário.

Principais Métodos da Classe TCustomADODataset

Apresenta-se a seguir uma descrição dos principais métodos da classe TCustomADODataset, além daqueles herdados das suas classes-base.

ApplyUpdates

Declaração

```
function BookmarkValid(Bookmark: TBookmark): Boolean; override;
```

Esse método determina se o bookmark passado como parâmetro é válido, isto é, se há um valor atribuído ao mesmo.

CancelUpdates

Declaração

```
procedure CancelUpdates;
```

Esse método cancela as alterações feitas localmente, para um componente que acessa uma tabela.

Clone

Declaração

```
procedure Clone(Source: TCustomADODataSet; LockType: TLockType = ltUnspecified);
```

Esse método copia os registros de outro componente derivado de TCustomADODataset passado como parâmetro.

CompareBookmarks

Declaração

```
function CompareBookmarks(Bookmark1, Bookmark2: TBookmark): Integer; override;
```

Este método compara dois objetos da classe TBookmark, retornando a diferença entre os dois Bookmarks.

DeleteRecords

Declaração

```
procedure DeleteRecords(AffectRecords: TAffectRecords = arAll);
```

Este método permite a remoção de um ou mais registros acessados pelo componente, dependendo do valor passado pelo parâmetro AffectRecords passado como parâmetro. Este parâmetro pode assumir um dos seguintes valores:

- arCurrent: Remove apenas o registro corrente.
- arFiltered: Remove apenas os registros que satisfazem a condição definida pelo filtro aplicado a um componente.
- arAll: Remove todos os registros acessados pelo componente.

DeleteRecords

Declaração

```
procedure FilterOnBookmarks(Bookmarks: array of const);
```

Este método aplica um filtro que faz com que apenas sejam exibidos os registros para os quais foi definido um Bookmark.

LoadFromFile

Declaração

```
procedure LoadFromFile(const FileName: WideString);
```

Este método recupera de um arquivo (cujo nome é passado como primeiro parâmetro na forma de uma string) os registros acessados pelo componente.

Requery

Declaração

```
procedure Requery(Options: TExecuteOptions = []);
```

Este método atualiza a exibição dos registros acessados pelo componente, em função das opções definidas pelo parâmetro Options, que pode ter um dos valores a seguir:

- eoAsyncExecute: O comando de atualização de registros é executado de forma assíncrona.
- eoAsyncFetch: O comando de atualização de registros é executado de forma assíncrona, dependendo do valor especificado na propriedade Cache.
- eoAsyncFetchNonBlocking: O comando de atualização de registros é executado sem bloquear a thread corrente.
- eoExecuteNoRecords: Aplica-se a comandos ou stored procedures que não retornam um conjunto de registros.

SaveToFile

Declaração

```
procedure SaveToFile(const FileName: String = ''; Format: TPersistFormat = pfADTG);
```

Este método salva em um arquivo (cujo nome é passado como primeiro parâmetro na forma de uma string) os registros acessados pelo componente, no formato definido pelo segundo parâmetro.

Os formatos possíveis para o segundo parâmetro são:

- pfADTG: Formato Advanced Data Tablegram
- pfXML: Formato XML

Seek

Declaração

```
function Seek(const KeyValues: Variant; SeekOption: TSeekOption = soFirstEQ): Boolean;
```

Este método pesquisa os registros com base em valores a serem armazenados nos campos que definem o índice corrente.

O primeiro parâmetro, na forma de uma array de Variants, contém os valores a serem pesquisados nos campos que formam o índice corrente.

O segundo parâmetro define como deverá ser feita a pesquisa, e pode ter um dos seguintes valores:

- soFirstEQ: Se um ou mais registros possuem nos campos que formam o indice corrente os valores definidos na pesquisa, o primeiro registro deste conjunto é definido como o registro corrente. Se nenhum registro atende aos requisitos especificados, o último registro acessado pelo componente é definido como o registro corrente.
- soLastEQ: Se um ou mais registros possuem nos campos que formam o indice corrente os valores definidos na pesquisa, o último registro deste conjunto é definido como o registro corrente. Se nenhum registro atende aos requisitos especificados, o último registro acessado pelo componente é definido como o registro corrente.
- .soAfterEQ: Se um ou mais registros possuem nos campos que formam o indice corrente os valores definidos na pesquisa, o registro após o primeiro registro deste conjunto é definido como o registro corrente. Se nenhum registro atende aos requisitos especificados, o último registro acessado pelo componente é definido como o registro corrente.
- soAfter: Se um ou mais registros possuem nos campos que formam o indice corrente os valores definidos na pesquisa, o registro após o último registro deste conjunto é definido como o registro corrente. Se nenhum registro atende aos requisitos especificados, o último registro acessado pelo componente é definido como o registro corrente.
- soBeforeEQ: Se um ou mais registros possuem nos campos que formam o indice corrente os valores definidos na pesquisa, o registro imediatamente anterior ao primeiro registro deste conjunto é definido como o registro corrente. Se nenhum registro atende aos requisitos especificados, o último registro acessado pelo componente é definido como o registro corrente.

- soBefore: Se um ou mais registros possuem nos campos que formam o índice corrente os valores definidos na pesquisa, o registro imediatamente anterior ao primeiro registro deste conjunto é definido como o registro corrente. Se nenhum registro atende aos requisitos especificados, o último registro acessado pelo componente é definido como o registro corrente.

Principais Eventos da Classe TCustomADODataset

Apresenta-se a seguir uma descrição dos principais eventos da classe TCustomADODataset, além daqueles herdados das suas classes-base.

OnEndOfRecordSet

Este evento ocorre quando o último registro acessado pelo componente é o registro corrente.

OnFetchComplete

Este evento ocorre quando o componente termina de carregar o conjunto de registros por ele acessado.

OnFetchProgress

Este evento ocorre enquanto o componente carrega o conjunto de registros por ele acessado.

OnFieldChangeComplete

Este evento ocorre após a alteração do valor armazenado em um campo do registro corrente.

OnMoveComplete

Este evento ocorre após a alteração de posição do registro definido como registro corrente.

OnRecordChangeComplete

Este evento ocorre após a alteração dos valores armazenados nos campos de diversos registros acessados pelo componente.

OnRecordsetChangeComplete

Este evento ocorre após a alteração do conjunto de registros acessados pelo componente.

OnWillChangeField

Este evento ocorre antes da alteração do valor armazenado em um campo do registro corrente.

OnWillMove

Este evento ocorre antes da alteração de posição do registro definido como registro corrente.

OnWillChangeRecord

Este evento ocorre antes da alteração dos valores armazenados nos campos de diversos registros acessados pelo componente.

OnWillChangeRecordset

Este evento ocorre antes da alteração do conjunto de registros acessados pelo componente.

A Classe TADOCommand

Esta classe é derivada diretamente da classe TComponent, e é implementada na forma de um componente (o segundo componente da página ADO da paleta de componentes).

A classe TADOCommand permite que se acesse diretamente um banco de dados ADO via declarações SQL que não retornem um conjunto de registros a serem editados na aplicação.

Principais Propriedades da Classe TADOCommand

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TADOCommand, além daquelas herdadas das suas classes ancestrais

CommandObject

Essa propriedade é definida como um objeto que acessa diretamente o objeto Activex responsável pela conexão.

CommandText

Essa propriedade é definida como uma variável do tipo WideString, e define o comando a ser executado (geralmente na forma de uma declaração SQL).

CommandTimeOut

Essa propriedade é definida como uma variável inteira, e define o tempo máximo (em segundos) para a execução de um comando (senão considera-se que o conando não foi executado com sucesso). Seu valor default é igual a 30.

CommandType

Essa propriedade é definida como uma variável do tipo TCommandType, e define o tipo de comando a ser executado.

Esta propriedade pode assumir um dos seguintes valores:

- CmdUnknown: O tipo de comando é desconhecido.
- cmdText: O comando é uma definição textual de uma declaração SQL ou chamada de uma Stored Procedure.

- cmdTable: O comando é, na realidade, o nome de uma tabela.
- cmdStoredProc: O comando é, na realidade, o nome de uma stored procedure.
- cmdFile: O comando se refere ao nome de um arquivo no qual um conjunto de registros está armazenado.
- cmdTableDirect: O comando é, na realidade, o nome de uma tabela, sendo todos os campos retornados.

As constantes cmdTable, cmdTableDirect, e cmdOpenFile não são usadas com o componente TADOCommand.

Connection

Essa propriedade é definida como uma variável do tipo TADOConnection, e define o tipo de conexão que será usada pelo componente.

ConnectionString

Essa propriedade é definida como uma variável do tipo WideString e define como será feita a conexão ao banco de dados.

As propriedades Connection e ConnectionString são mutuamente excludentes.

ParamCheck

Essa propriedade é definida como uma variável booleana, e define se a lista de parâmetros do componente deve ser gerada novamente quando a declaração SQL é alterada durante a execução do aplicativo.

Parameters

Essa propriedade é definida como um objeto da classe TParameters, e define os parâmetros usados em uma declaração SQL.

Prepared

Essa propriedade é definida como uma variável booleana, e define se a declaração SQL deve ser preparada (para otimização da sua execução) antes de ser executada.

PRINCIPAIS MÉTODOS DA CLASSE TADOCOMMAND

Apresenta-se a seguir uma descrição dos principais métodos da classe TADODataset, além daqueles herdados das suas classes-base.

Assign

Declaração

```
procedure Assign(Source: TPersistent); override;
```

Esse método copia as propriedades de outro componente TADOCommand para o componente corrente.

Cancel

Declaração

```
procedure Cancel;
```

Esse método suspende a execução do comando definido na propriedade CommandText do componente.

Execute

Declaração

```
function Execute: _RecordSet; overload;
function Execute(const Parameters: OleVariant): _Recordset; overload;
function Execute(var RecordsAffected: Integer; var Parameters: OleVariant; ExecuteOptions:
TExecuteOptions = []): _RecordSet; overload;
```

Esse método é responsável pela execução do comando especificado na propriedade CommandText do componente. Repare que este método é sobrecarregado, podendo ser executado com diferentes conjuntos de parâmetros.

Principais Eventos da Classe TADOCommand

Esta classe não possui eventos.

A Classe TADODataset

Esta classe é derivada diretamente da classe TCustomADODataset, e é implementada na forma de um componente (o terceiro componente da página ADO da paleta de componentes).

A classe TADODataSet, derivada por herança direta da classe TCustomADODataSet, incorpora as principais características do mecanismo ADO a alguns dos métodos declarados na classe TDataSet, sobrecarregando-os (no caso de métodos virtuais) ou implementando-os (no caso de métodos abstratos).

Os métodos implementados por essa classe não são normalmente usados no desenvolvimento de aplicações em Delphi, pois geralmente são utilizados componentes representados por classes derivadas por herança desta classe (TADOTable, TADOQuery e TADOStoredProc).

Principais Propriedades da Classe TADODataSet

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TADODataSet, além daquelas herdadas das suas classes ancestrais.

RDSConnection

Permite que a conexão seja feita a um banco de dados remoto, através de um componente TRDSConnection, cujo nome é especificado nesta propriedade.

Principais Métodos da Classe TADODataset

Apresenta-se a seguir uma descrição dos principais métodos da classe TADODataset, além daqueles herdados das suas classes-base.

CreateDataset

Declaração

```
function Createdataset;
```

Esse método é responsável pela obtenção dos registros a serem acessados pelo componente, sendo usado internamente e não devendo ser chamado diretamente pela aplicação.

GetIndexNames

Declaração

```
procedure GetIndexNames(List: TStrings);
```

Esse método retorna, na lista de strings passada como parâmetro na forma de um objeto da classe TStrings, os nomes dos índices definidos para a tabela.

Principais Eventos da Classe TADODataset

Esta classe não Implementa novos eventos, utilizando os herdados das suas classes-base.

KNOW-HOW em: Classes de Acesso Direto a Bancos de dados via BDE – As Classes TADOTable e TADOQuery

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 7.
- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi 7.
- Compreensão da classes TCustomADODataset, TADODataset e TADOConnection.

METODOLOGIA

- Apresentação das classes e dos componentes de acesso direto a bancos de dados via ADO, juntamente com uma descrição das suas propriedades, métodos e eventos.

TÉCNICA

- Descrição das classes e dos componentes de acesso direto a bancos de dados e apresentação de exemplos de aplicação.
- Serão apresentadas neste tópico as classes usadas para acesso direto a bancos de dados via ADO – as classes TADOTable e TADOQuery.

A Classe TADOTable

A classe TADOTable é derivada por herança direta da classe TADODataSet, sendo normalmente utilizada para se estabelecer uma conexão a uma tabela individual do banco de dados.

Principais Propriedades da Classe TADOTable

Apresenta-se a seguir uma descrição das principais propriedades da classe TADOTable, além daquelas herdadas das suas classes ancestrais.

IndexFieldNames

Essa propriedade é definida como uma variável do tipo string, que armazena os nomes dos campos que compõem o índice corrente, separados por um ponto-e-vírgula.

As propriedades IndexName e IndexFieldNames são mutuamente excludentes, isto é, a definição de um valor para uma propriedade anula o valor definido para a outra.

IndexName

Essa propriedade é definida como uma variável do tipo string e define o nome do índice corrente da tabela.

MasterFields

Essa propriedade é definida como uma variável do tipo string e define os nomes dos campos da tabela principal em um relacionamento (separados por um ponto-e-vírgula), devendo ser definida no componente que representa a tabela secundária.

MasterSource

Essa propriedade é definida como um objeto da classe TDataSource e define o nome do componente DataSource ao qual está associado o componente da classe TTable que representa a tabela principal em um relacionamento entre tabelas. Essa propriedade deve ser definida apenas no componente que representa a tabela secundária no relacionamento.

ReadOnly

Essa propriedade é definida como uma variável booleana e define se a tabela pode ser acessada apenas para visualização de registros.

TableDirect

Essa propriedade é definida como uma variável booleana, e define se o acesso à tabela é feito de forma direta, ou se é necessária a criação de uma declaração SQL em background.

TableName

Essa propriedade é definida como uma variável do tipo string e define os nomes da tabela representada pelo componente.

Principais Métodos da Classe TADOTable

Apresenta-se a seguir uma descrição dos principais métodos da classe TADOTable, além daqueles herdados das suas classes ancestrais:

GetIndexNames

Declaração

```
procedure GetIndexNames(List: TStrings);
```

Esse método armazena em um objeto da classe TStrings (passado como parâmetro na chamada do método) os nomes dos índices definidos para a tabela representada pelo componente.

Principais Eventos da Classe TADOTable

Esta classe não implementa novos eventos, utilizando os herdados das suas classes-base.

A Classe TADOQuery

A classe TADOQuery é derivada por herança direta da classe TCustomADODataSet, sendo normalmente utilizada para se estabelecer uma conexão a uma ou mais tabelas de um banco de dados acessadas via ADO usando-se declarações SQL.

Principais Propriedades da Classe TADOQuery

Apresenta-se a seguir uma descrição das principais propriedades da classe TADOQuery, além daquelas herdadas das suas classes ancestrais:

RowsAffected

Essa propriedade é definida como uma variável inteira e informa a quantidade de registros afetados pela última declaração SQL executada pelo componente.

SQL

Essa propriedade é definida como um objeto da classe TStrings (que é uma lista de strings) na qual deve ser armazenada a declaração SQL a ser executada mediante uma chamada aos métodos Open ou ExecSQL do componente.

Principais Métodos da Classe TADOQuery

Apresenta-se a seguir uma descrição dos principais métodos da classe TADOQuery, além daqueles herdados das suas classes ancestrais.

ExecSQL

Declaração

```
procedure ExecSQL;
```

Esse método permite a execução de declarações SQL que envolvam a inserção, a remoção e a atualização de registros, isto é, declarações SQL que contêm as cláusulas Insert, Delete e Update.

Declarações SQL que envolvem apenas consultas resultantes da utilização da cláusula SELECT devem ser executadas mediante uma chamada ao método Open do componente.

Principais Eventos da Classe TADOQuery

Esta classe não implementa novos eventos, utilizando os herdados das suas classes-base.

Exemplos de Aplicação

Estabelecendo uma Conexão a Bancos de Dados do MS Acess com o Componente ADOConnection

Para se conectar a um banco de dados do MS Access com o componente ADOConnection, você deve executar os seguintes procedimentos:

1. Coloque um componente ADOConnection no formulário ou Datamodule.

2. Selecione os três pontinhos à direita da sua propriedade ConnectionString, para exibir a caixa de diálogo mostrada na figura a seguir:

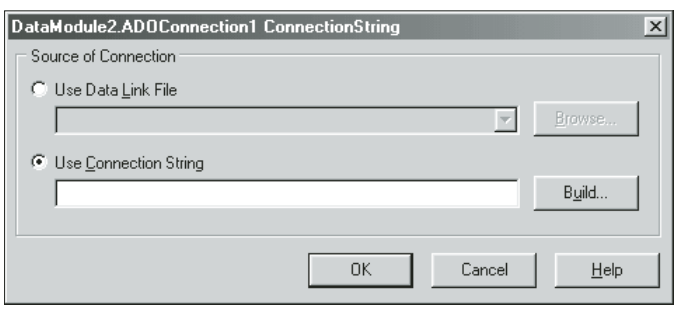


Figura 24.1: A caixa de diálogo Connection String.

3. Selecione o botão Build para exibir a caixa de diálogo de Propriedades de vinculação de dados, mostrada na figura a seguir, e cuja finalidade é nos auxiliar na criação da string de conexão.

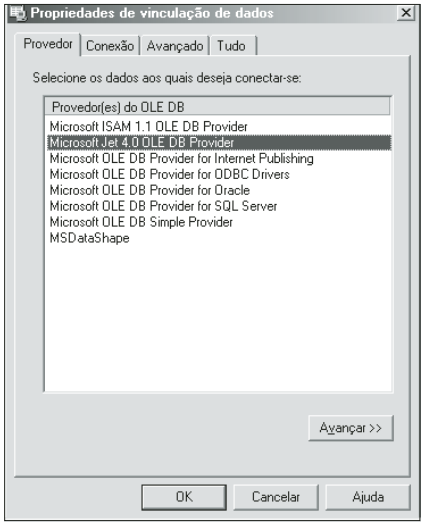


Figura 24.2: A caixa de diálogo de Propriedades de vinculação de dados.

4. Nesta caixa de diálogo, selecione o item Microsoft Jet 4.0 OLE DB Provider, como indicado na figura anterior.

5. Selecione o botão Next, para exibir a próxima página desta caixa de diálogo. Nesta página, você deve informar o nome do banco de dados ou localizar o arquivo no qual o mesmo se encontra armazenado. Neste caso, basta selecionar o botão à direita da caixa de texto "Insira o nome de um banco de dados" para exibir a caixa de diálogo padrão para seleção de arquivos do Windows (neste caso, para a seleção de arquivos mdb) e escolher o arquivo desejado, cujos nome e path completos aparecerão nesta caixa de texto, conforme mostrado na figura a seguir. Neste caso selecionou-se C:\Arquivos de programas\Arquivos comuns\Borland Shared\Data\dbdemos.mdb

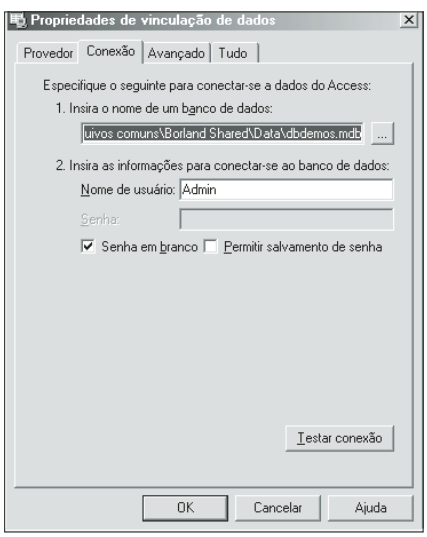


Figura 24.3: A página Conexão da caixa de diálogo de Propriedades de vinculação de dados.

6. Selecione o botão Testar Conexão, para verificar se a execução foi efetuada corretamente. Em caso positivo, será exibida a caixa de diálogo mostrada na figura a seguir.

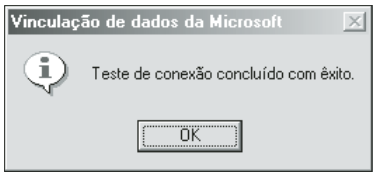


Figura 24.4: Testando a conexão.

7. Selecione Ok para fechar a caixa de diálogo de Propriedades de vinculação de dados e verificar a string de conexão na caixa de diálogo String Connection, como mostrado na próxima figura. Repare que a string de conexão é bastante complexa, embora possa ser definida manualmente é bem mais simples utilizar a caixa de diálogo Data Link Properties.

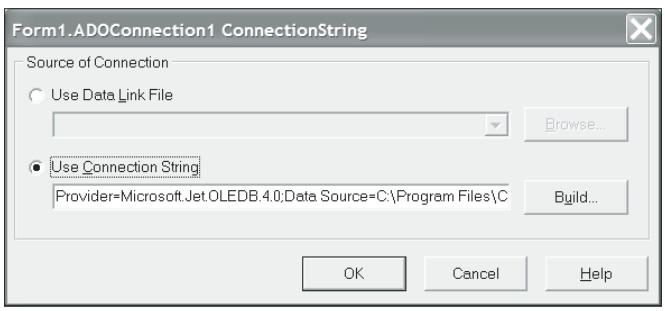


Figura 24.5: A caixa de diálogo Connection String.

8. Selecione Ok para fechar a caixa de diálogo Connection String e defina como True o valor da propriedade Connected deste componente. Será exibida a caixa de diálogo Database Login. Mantenha os valores de UserName e password em branco e selecione o botão Ok para efetuar a conexão. Se você definir como False o valor da propriedade LoginPrompt, esta caixa de diálogo não será exibida (e neste caso não é necessária – em outras situações será necessário fornecer tais valores a partir do procedimento associado ao seu evento OnLogin).

Repare o tamanho da string usada para definir a conexão:

```
Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;Data Source=C:\Arquivos de
programas\Arquivos comuns\Borland Shared\Data\dbdemos.mdb;Mode=Share Deny
None;Extended Properties="";Jet OLEDB:System database="";Jet OLEDB:Registry
Path="";Jet OLEDB:Database Password="";Jet OLEDB:Engine Type=5;Jet OLEDB:Database
Locking Mode=1;Jet OLEDB:Global Partial Bulk Ops=2;Jet OLEDB:Global Bulk
Transactions=1;Jet OLEDB:New Database Password="";Jet OLEDB:Create System
Database=False;Jet OLEDB:Encrypt Database=False;Jet OLEDB:Don't Copy Locale on
Compact=False;Jet OLEDB:Compact Without Replica Repair=False;Jet OLEDB:SFP=False
```

Embora os componentes ADOCommand, ADODataset, ADOStoredproc, ADOTable e ADOQuery possam acessar o banco de dados através de sua própria propriedade Connection string, recomenda-se que o façam através deste componente, para evitar o desperdício de memória e de tempo na abertura de várias conexões simultâneas.

Acessando tabelas do Access com o Componente ADOComando

Para utilizar o componente ADOCommand, você deve executar os seguintes procedimentos:

1. Coloque-o no mesmo datamodule em que foi colocado o componente ADOConnection (que chamei nestes exemplos de ADOConnectionTest) e atribua à sua propriedade Connection o nome do componente (que pode ser selecionado a partir da relação de componentes disponíveis).

2. Digite o código SQL a ser executado em sua propriedade CommandText.

3. Em um procedimento qualquer (como o de um evento OnClick de um botão de comando), execute o método Execute deste componente.

Repare que este componente, por não ser derivado da classe TDataset, não pode ter o seu nome diretamente atribuído à propriedade Dataset de um componente Datasource, de forma que os registros retornados em um comando Select possam ser visualizados diretamente em um dos componentes de visualização. Fica claro, portanto, que este componente é útil quando se quer executar declarações SQL sem a necessidade de visualizar os seus resultados.

Acessando Tabelas do Access com o Componente ADODataset

Para utilizar o componente ADODataset, você deve executar os seguintes procedimentos:

1. Coloque-o no mesmo Datamodule em que foi colocado o componente ADOConnection (que chamei nestes exemplos de ADOConnectionTest) e atribua à sua propriedade Connection o nome do componente (que pode ser selecionado a partir da relação de componentes disponíveis).

2. Digite o código SQL a ser executado em sua propriedade CommandText, como por exemplo:

```
Select * From Customer
```

3. Atribua o valor True a sua propriedade Active.

Repare que este componente, por ser derivado da classe TDataset, pode ter o seu nome diretamente atribuído à propriedade Dataset de um componente Datasource, de forma que os registros retornados em um comando Select possam ser visualizados diretamente em um dos componentes de visualização. É útil quando se quer acessar dados a partir de um banco de dados remoto, pois possui a propriedade RDSConnection (à qual pode ser atribuído o nome de um componente TRDSConnection) – o que não pode ser feito com os componentes TADOQuery e TADOTable.

Este componente também possui um editor de campos (Fields Editor) semelhante ao existente para os componentes TTable, TQuery e seus equivalentes TADOTable e TADOQuery.

A figura a seguir mostra um formulário em que os registros obtidos a partir deste componente são visualizados em um DBGrid.

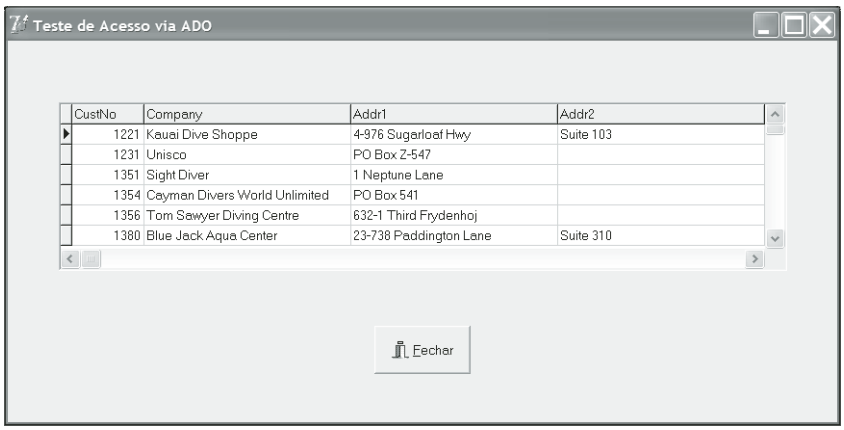


Figura 24.6: Acessando uma tabela com o componente ADODataset.

Como mostrado na figura a seguir, a propriedade CommandText pode ser editada a partir da caixa de diálogo Command Text Editor, mostrada na figura a seguir.

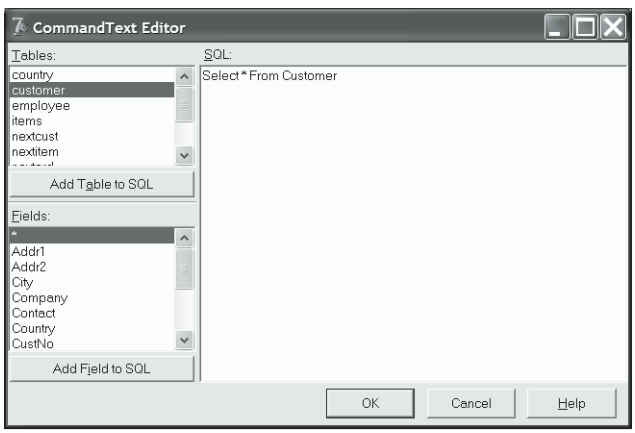


Figura 24.7: Editando a propriedade CommandText com a caixa de diálogo CommandText Editor.

Acessando tabelas do Access com o Componente ADOTable

Para utilizar o componente ADOTable, você deve executar os seguintes procedimentos:

1. Coloque-o no mesmo Datamodule em que foi colocado o componente ADOConnection (que chamei nestes exemplos de ADOConnectionTest) e atribua à sua propriedade Connection o nome do componente (que pode ser selecionado a partir da relação de componentes disponíveis).

2. Selecione a tabela desejada na propriedade TableName.

No mais, tudo se passa da mesma forma que com o componente Table, com o qual você já deve estar acostumado desde as primeiras versões do Delphi.

Repare que para este componente não existe a propriedade DatabaseName – em seu lugar temos as propriedades Connection e ConnectionString, que são mutuamente excludentes.

Acessando tabelas do Access com o Componente ADOQuery

Para utilizar o componente ADOQuery, você deve executar os seguintes procedimentos:

1. Coloque-o no mesmo datamodule em que foi colocado o componente ADOConnection (que chamei nestes exemplos de ADOConnectionTest) e atribua à sua propriedade Connection o nome do componente (que pode ser selecionado a partir da relação de componentes disponíveis).

2. Digite o código a ser executado na sua propriedade SQL.

No mais, tudo se passa da mesma forma que com o componente TQuery, com o qual você já deve estar acostumado desde as primeiras versões do Delphi

Repare que para este componente também não existe a propriedade DatabaseName - em seu lugar temos as propriedades Connection e ConnectionString, que são umtuamente excludentes.

Diferenças na Utilização dos Componentes Table x ADOTable, e Query x ADOQuery

Quase tudo que é feito com os componentes Table e Query pode ser feito pelos componentes ADOTable e ADOQuery, respectivamente. Nos próximos tópicos, abordaremos apenas algumas características não suportadas e que diferenciam a utilização destes componentes.

Definição do Índice Corrente

No Delphi, a definição do índice corrente de uma tabela acessada usando-se um componente Table pode ser feita da seguinte maneira:

Na fase de projeto: Definindo-se, diretamente no Object Inspector, o valor da propriedade IndexName do componente Table que representa a tabela (se essa propriedade não for definida, será utilizada a chave primária) ou definindo-se, diretamente no Object Inspector, o valor da propriedade IndexFieldNames do componente Table que representa a tabela.

Conforme já foi descrito anteriormente, as propriedades IndexName e IndexFieldNames são mutuamente excludentes. Atribuir um valor a uma propriedade anula o valor definido para a outra.

Durante a execução do aplicativo: Definindo-se via código, mediante uma operação de atribuição, o valor das propriedades descritas no tópico anterior. No caso da propriedade IndexFieldNames, os nomes dos campos que formam o índice devem vir separados por ponto-e-vírgula. No caso do componente ADOTable, no entanto, você deve atribuir o valor False à sua propriedade Active amates de alterar o índice corrente.

Pesquisando Registros em Tabelas Representadas Pelo Componente ADOTable

Diferentemente do componente Table, o componente ADOTable não possui os métodos FindKey e FindNearest, mas apenas o método Locate, que permite a busca exata ou aproximada de um registro, por campos que não façam parte do índice corrente da tabela.

Criação de Tabelas em Run-time

O componente TADOTable não possui um método semelhante ao método CreateTable do componente Table, que permita a criação de tabelas em run-time.

Componentes e Métodos de Navegação

Os mesmos métodos de navegação são suportados pelos dois conjuntos de componentes – BDE e ADO.

Utilização de Parâmetros em Declarações SQL

No caso do componente ADOQuery, o parâmetro não é diretamente reconhecido a partir da declaração SQL. Neste caso, você deverá completar as informações do parâmetro no Object Inspector, como indicado na figura a seguir.

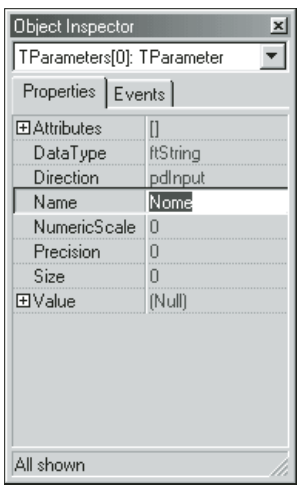


Figura 24.8: Definindo o parâmetro da declaração SQL.

Considere, neste exemplo, a seguinte declaração SQL (usada no exemplo do componente TQuery):

```
Select * From Country Where Continent =:Nome
```

A redefinição do parâmetro e atualização da query é feita no procedimento associado ao evento OnChange do comboBox. Neste caso, no entanto, o código de definição do parâmetro deverá ser alterado, como mostrado a seguir:

```
procedure TFormParametros.ComboBox1Change(Sender: TObject);

begin
     ADOQuery1.Close;
     ADOQuery1.Parameters.ParamValues['Nome'] := ComboBox1.Items[ComboBox1.ItemIndex];
     ADOQuery1.Open;
end;
```

Repare que o componente ADOQuery não possui o método ParambyName – ao invés disso deve ser usada a propriedade Parameters que, por sua vez, possui uma array de parâmetros chamada ParamValues, na qual os parâmetros podem ser acessados pelo seu nome.

Além disso, deve ser notada a inexistência do método Prepare, e que o componente não pode ter o valor da sua propriedade Active como True enquanto não for definido um valor para o parâmetro. Isto pode ser feito no procedimento associado ao evento OnCreate do formulário ou no próprio Object Inspector, como mostrado na figura a seguir.

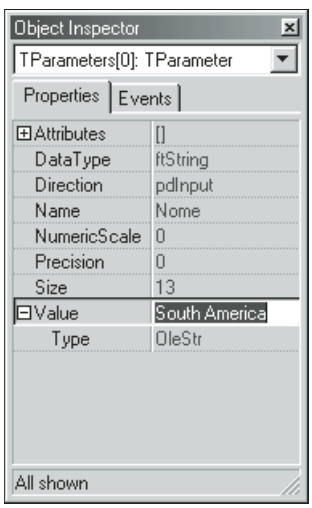


Figura 24.9: Definindo um valor para o parâmetro da declaração SQL.

Capítulo 25

Banco de Dados – Componentes de Acesso via DBExpress

Neste capítulo serão discutidos em maiores detalhes os componentes responsáveis pelo acesso a dados via DBExpress, alguns já apresentados na primeira parte do livro, a partir de uma aplicação desenvolvida com o Delphi 7. Conforme já foi descrito anteriormente, o DBExpress foi especialmente desenvolvido pela Borland para permitir o desenvolvimento de aplicações multiplataforma – capazes de ser compiladas para o ambiente Windows (com o Delphi) e Linux (com o Kylix) em aplicações desenvolvidas com base na CLX.

KNOW-HOW em: Componentes de Acesso a Bancos de Dados via DBExpress

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 7.
- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi 7.
- Compreensão da classe TDataset.

METODOLOGIA

- Apresentação dos componentes de acesso a bancos de dados via DBExpress, juntamente com uma descrição das suas propriedades, métodos e eventos.

TÉCNICA

- Descrição dos componentes de acesso a bancos de dados via DBExpress.

Este mecanismo foi bastante empregado na primeira parte do livro, e sua principal característica reside no fato de ser um mecanismo unidirecional de acesso a dados, o que significa que você não pode navegar diretamente pelos registros acessados através de seus componentes, como ocorria com os componentes de acesso via BDE e ADO. Uma alternativa, neste caso, consiste em utilizar o componente ClientDataset em conjunto com um DatasetProvider e o componente de acesso ou, como já descrito anteriormente, o componente SimpleDataset, que reúne as funcionalidades de diversos componentes.

O Componente TSQLConnection

Este componente, derivado por herança da classe TCustomConnection, permite que se estabeleça uma conexão a um banco de dados via DBExpress. Este componente tem uma funcionalidade semelhante ao TDatabase no mecanismo Borland Database Engine e ao ADOConnection do ADO, e permite que diversos componentes de acesso estejam diretamente conectados a ele.

Principais Propriedades da Classe TSQLConnection

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TADOConnection, além daquelas herdadas das suas classes-base:

DriverName

Essa propriedade é definida como uma variável do tipo string, e define o driver para o tipo de banco de dados que será acessado. Por enquanto, estão disponíveis drivers para INTERBASE, MYSQL, ORACLE, MSSQL, INFORMIX e DB2.

ConnectionName

Essa propriedade é definida como uma variável do tipo string, e define o nome da conexão representada pelo componente. É similar à propriedade DatabaseName do componente Database do BDE.

Connected

Essa propriedade é definida como uma variável booleana, e define se uma conexão foi ou não estabelecida pelo componente.

LoginPrompt

Essa propriedade é definida como uma variável booleana, e define se uma caixa de diálogo de Login deve ou não ser exibida ao se tentar estabelecer uma conexão através deste componente.

KeepConnection

Essa propriedade é definida como uma variável booleana, e define se a conexão deve continuar permanentemente ativa, mesmo que não se esteja efetuando nenhum acesso aos dados.

Name

Essa propriedade é definida como uma variável do tipo string, e define o nome pelo qual o componente será referenciado no código da aplicação.

A maioria destas propriedades pode ser configurada usando-se a caixa de diálogo DBExpress Connection, mostrada na figura a seguir. Esta caixa de diálogo é exibida dando-se um duplo clique sobre o componente ou selecionando-se o item Connection Editor do seu menu pop-up, e também pode ser usada para armazenar os valores default do username e password.

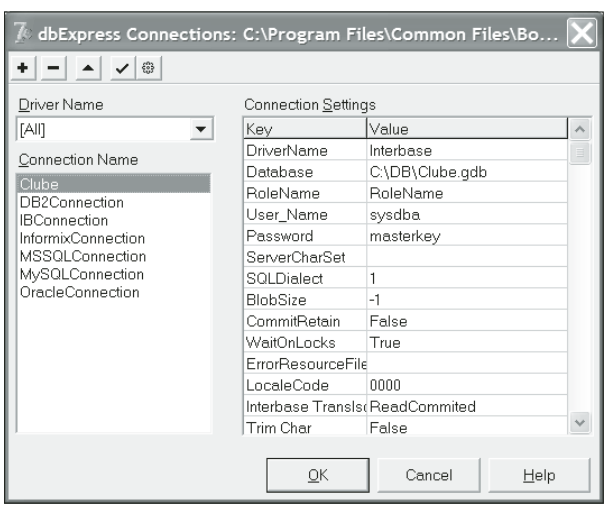


Figura 25.1: A caixa de diálogo DBExpress Connections.

Algumas destas propriedades podem ser configuradas usando-se a caixa de diálogo Value List Editor, exibida através da seleção da propriedade Params.

Principais Métodos da Classe TSQLConnection

Apresenta-se a seguir uma descrição dos principais métodos da classe TSQLConnection, além daquelas herdadas das suas classes-base.

CloseAllDataSets

Declaração

```
procedure CloseDataSets;
```

Esse método desativa todos os objetos derivados da classe TDataset a ele conctados, sem no entanto finalizar a sua própria conexão ao banco de dados.

Commit

Declaração

```
procedure Commit;
```

Esse método realiza um commit sobre a transação corrente.

Execute

Declaração

```
function Execute(const SQL: string; Params: TParams
; ResultSet:Pointer=nil): Integer;
```

Esse método executa um comando SQL sobre o servidor de banco de dados. Repare que este método é sobrecarregado, possuindo duas implementações distintas.

O comando a ser executado é passado na forma de uma string, além de outros parâmetros que indicam como os registros devem ser afetados.

GetProcedureNames

Declaração

```
procedure GetProcedureNames(List: TStrings);
```

Esse método armazena na lista de strings passada como parâmetro os nomes de todos os procedimentos armazenados existentes no banco de dados ao qual o componente está conectado.

GetTableNames

Declaração

```
procedure GetTableNames(List: TStrings; SystemTables: Boolean = False);
```

Esse método armazena na lista de strings passada como parâmetro os nomes de todas as tabelas existentes no banco de dados ao qual o componente está conectado.

Open

Declaração

```
procedure Open;
```

Esse método ativa uma conexão ao banco de dados ao qual o componente está conectado.

Rollback

Declaração

```
procedure Rollback(TransDesc: TTransactionDesc);
```

Esse método realiza um rollback sobre a transação corrente.

Principais Eventos da Classe TSQLConnection

Apresenta-se a seguir uma descrição dos principais eventos da classe TSQLConnection, além daquelas herdadas das suas classes-base.

AfterConnect

Este evento ocorre logo após uma conexão ser estabelecida pelo componente.

AfterDisConnect

Este evento ocorre logo após uma conexão ser finalizada pelo componente.

BeforeConnect

Este evento ocorre imediatamente antes de uma conexão ser estabelecida pelo componente.

BeforeDisConnect

Este evento ocorre imediatamente antes de uma conexão ser finalizada pelo componente.

OnLogin

Este evento ocorre imediatamente após o evento BeforeConnect, e permite que se definam valores para o username e o password de acesso ao banco de dados.

A Classe TCustomSQLDataset

Conforme descrito anteriormente, a classe TDataSet implementa a funcionalidade genérica para acesso a tabelas, sem incorporar as funções de qualquer API.

A classe TCustomSQLDataSet, derivada por herança direta da classe TDataSet, incorpora aa principais características do mecanismo DBExpress a alguns dos métodos declarados na classe TDataSet, sobrecarregando-os (no caso de métodos virtuais) ou implementando-os (no caso de métodos abstratos).

Os métodos implementados por essa classe não são normalmente usados no desenvolvimento de aplicações em Delphi, pois geralmente são utilizados componentes representados por classes derivadas por herança desta classe.

Principais Propriedades da Classe TCustomSQLDataSet

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TDataSet.

SQLConnection

Essa propriedade é definida como uma variável do tipo TSQLConnection, e define o nome do componente SQLConnection através do qual será feita a conexão ao servidor do banco de dados.

CommandType

Essa propriedade é definida como uma variável do tipo TSQLCommandType, e define se o acesso será feito diretamente a uma tabela (se seu valor for ctTable), via declaração SQL (se seu valor for ctQuery) ou a um procedimento armazenado (se seu valor for igual a ctStoredProc).

CommandText

Essa propriedade é definida como uma variável do tipo string, e seu valor vai depender do valor previamente atribuído à propriedade CommandType.

Se CommandType for igual a ctTable, esta propriedade deverá armazenar o nome de uma tabela.

Se CommandType for igual a ctQuery, esta propriedade deverá armazenar um comando SQL.

Se CommandType for igual a ctStoredProc, esta propriedade deverá armazenar o nome de um procedimento armazenado.

Principais Eventos da Classe TCustomSQLDataset

Este componente não implementa nenhum método, possuindo apenas os herdados de sua classe-base.

A Classe TSQLDataSet

Esta classe é derivada diretamente da classe TCustomSQLDataset, e é implementada na forma de um componente.

A classe TSQLDataSet permite que se acesse diretamente uma tabela de um banco de dados, um conjunto de tabelas via declarações SQL ou um procedimento armazenado, dependendo do valor atribuído a sua propriedade CommandType.

Principais Propriedades da Classe TSQLDataSet

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TSQLDataSet, além daquelas herdadas das suas classes ancestrais.

Active

Esta propriedade é definida como uma variável booleana que define se a conexão representada pelo componente está ou não ativa.

SQLConnection

Essa propriedade é definida como uma variável do tipo TSQLConnection, e define o nome do componente SQLConnection através do qual será feita a conexão ao servidor do banco de dados.

CommandType

Essa propriedade é definida como uma variável do tipo TSQLCommandType, e define se o acesso será feito diretamente a uma tabela (se seu valor for ctTable), via declaração SQL (se seu valor for ctQuery) ou a um procedimento armazenado (se seu valor for igual a ctStoredProc).

CommandText

Essa propriedade é definida como uma variável do tipo string, e seu valor vai depender do valor previamente atribuído à propriedade CommandType.

Se CommandType for igual a ctTable, esta propriedade deverá armazenar o nome de uma tabela.

Se CommandType for igual a ctQuery, esta propriedade deverá armazenar um comando SQL.

Se CommandType for igual a ctStoredProc, esta propriedade deverá armazenar o nome de um procedimento armazenado.

KNOW-HOW em: Classes de Acesso Direto a Bancos de Dados via DBExpress – As Classes TSQLTable, TSQLQuery e TSQLStoredProc

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 6 e o Kylix.
- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi 7.
- Compreensão das classes TCustomSQLDataset, TSQLDataset e TSQLConnection.

METODOLOGIA

- Apresentação das classes e dos componentes de acesso direto a bancos de dados via DBExpress, juntamente com uma descrição das suas propriedades, métodos e eventos.

TÉCNICA

- Descrição das classes e dos componentes de acesso direto a bancos de dados e apresentação de exemplos de aplicação.

Serão apresentadas neste tópico as classes usadas para acesso direto a bancos de dados via DBExpress – as classes TSQLTable, TSQLQuery e TSQLStoredProc.

A CLASSE TSQLTABLE

A classe TSQLTable é derivada por herança direta da classe TSQLDataSet, sendo normalmente utilizada para se estabelecer uma conexão a uma tabela individual do banco de dados. Trata-se de uma especialização da classe TSQLDataSet com CommandType igual a ctTable.

PRINCIPAIS PROPRIEDADES DA CLASSE TSQLTABLE

Apresenta-se a seguir uma descrição das principais propriedades da classe TSQLTable, além daquelas herdadas das suas classes ancestrais.

Active

Esta propriedade é definida como uma variável booleana que define se a conexão representada pelo componente está ou não ativa.

IndexFieldNames

Essa propriedade é definida como uma variável do tipo string, que armazena os nomes dos campos que compõem o índice corrente, separados por um ponto-e-vírgula.

As propriedades IndexName e IndexFieldNames são mutuamente excludentes, isto é, a definição de um valor para uma propriedade anula o valor definido para a outra.

IndexName

Essa propriedade é definida como uma variável do tipo string e define o nome do índice corrente da tabela.

MasterFields

Essa propriedade é definida como uma variável do tipo string e define os nomes dos campos da tabela principal em um relacionamento (separados por um ponto-e-vírgula), devendo ser definida no componente que representa a tabela secundária.

MasterSource

Essa propriedade é definida como um objeto da classe TDataSource e define o nome do componente DataSource ao qual está associado o componente da classe TSQLTable que representa a tabela principal em um relacionamento entre tabelas. Essa propriedade deve ser definida apenas no componente que representa a tabela secundária no relacionamento.

TableName

Essa propriedade é definida como uma variável do tipo string e define os nomes da tabela representada pelo componente.

Principais Métodos da Classe TSQLTable

Apresenta-se a seguir uma descrição dos principais métodos da classe TSQLTable, além daqueles herdados das suas classes ancestrais:

GetIndexNames

Declaração

```
procedure GetIndexNames(List: TStrings);
```

Esse método armazena em um objeto da classe TStrings (passado como parâmetro na chamada do método) os nomes dos índices definidos para a tabela representada pelo componente.

Principais Eventos da Classe TSQLTable

Esta classe não implementa novos eventos, utilizando os herdados das suas classes-base.

A Classe TSQLQuery

A classe TSQLQuery é derivada por herança direta da classe TSQLDataSet, sendo normalmente utilizada para se estabelecer uma conexão a uma ou mais tabelas de um banco de dados acessadas via DBExpress usando-se declarações SQL. Trata-se de uma especialização da classe TSQLDataSet com CommandType igual a ctQuery.

Principais Propriedades da Classe TSQLQuery

Apresenta-se a seguir uma descrição das principais propriedades da classe TSQLQuery, além daquelas herdadas das suas classes ancestrais:

Active

Esta propriedade é definida como uma variável booleana que define se a conexão representada pelo componente está ou não ativa.

SQL

Essa propriedade é definida como um objeto da classe TStrings (que é uma lista de strings) na qual deve ser armazenada a declaração SQL a ser executada mediante uma chamada aos métodos Open ou ExecSQL do componente.

Principais Métodos da Classe TSQLQuery

Apresenta-se a seguir uma descrição dos principais métodos da classe TSQLQuery, além daquelas herdadas das suas classes ancestrais.

ExecSQL

Declaração

```
function ExecSQL(ExecDirect: Boolean = False): Integer; override;
```

Esse método permite a execução de declarações SQL que envolvam a inserção, a remoção e a atualização de registros, isto é, declarações SQL que contêm as cláusulas Insert, Delete e Update.

Declarações SQL que envolvem apenas consultas resultantes da utilização da cláusula SELECT devem ser executadas mediante uma chamada ao método Open do componente.

Principais Eventos da Classe TSQLQuery

Esta classe não implementa novos eventos, utilizando os herdados das suas classes-base.

A Classe TSimpleDataset

Conforme descrito anteriormente, os componentes que acessam bancos de dados via DBExpress só permitem acesso unidirecional ao banco de dados. Conseqüentemente, para contornar este problema seria necessário utilizar, em conjunto com cada um daqieles componentes, um componente DatasetProvider e um componente ClientDataset, responsável por editar localmente os registros, e permitir o seu acesso de forma bidirecional. Evidentemente, neste caso, as alterações seriam aplicadas ao banco de dados mediante uma chamada ao método ApplyUpdates do componente ClientDataset.

Este componente pode ser considerado como uma reunião de três componentes: Um SQLDataSet, um DatasetProvider e um ClientDataset. Conforme visto na primeira parte do livro, este componente simplifica bastante o desenvolvimento de aplicações que acessam bancos de dados via DBExpress.

Principais Propriedades da Classe TSimpleDataset

Apresenta-se a seguir uma descrição das principais propriedades da classe TSimpleDataset, além daquelas herdadas das suas classes ancestrais. Devido a sua importância, serão reapresentadas algumas das propriedades já definidas em seus componentes ancestrais.

Active

Esta propriedade é definida como uma variável booleana que define se a conexão representada pelo componente está ou não ativa.

DBConnection

Essa propriedade é definida como uma variável do tipo TSQLConnection, e define o nome do componente SQLConnection através do qual será feita a conexão ao servidor do banco de dados.

CommandType

Essa propriedade é definida como uma variável do tipo TSQLCommandType, e define se o acesso será feito diretamente a uma tabela (se seu valor for ctTable), via declaração SQL (se seu valor for ctQuery) ou a um procedimento armazenado (se seu valor for igual a ctStoredProc).

CommandText

Essa propriedade é definida como uma variável do tipo string, e seu valor vai depender do valor previamente atribuído à propriedade CommandType.

Se CommandType for igual a ctTable, esta propriedade deverá armazenar o nome de uma tabela.

Se CommandType for igual a ctQuery, esta propriedade deverá armazenar um comando SQL.

Se CommandType for igual a ctStoredProc, esta propriedade deverá armazenar o nome de um procedimento armazenado.

IndexFieldNames

Essa propriedade é definida como uma variável do tipo string, que armazena os nomes dos campos que compõem o índice corrente, separados por um ponto-e-vírgula.

IndexName

Essa propriedade é definida como uma variável do tipo string e define o nome do índice corrente da tabela.

MasterFields

Essa propriedade é definida como uma variável do tipo string e define os nomes dos campos da tabela principal em um relacionamento (separados por um ponto-e-vírgula), devendo ser definida no componente que representa a tabela secundária.

MasterSource

Essa propriedade é definida como um objeto da classe TDataSource e define o nome do componente DataSource ao qual está associado o componente da classe TSimpleDataset que representa a tabela principal em um relacionamento entre tabelas. Essa propriedade deve ser definida apenas no componente que representa a tabela secundária no relacionamento.

Principais Métodos da Classe TSimpleDataset

Esta classe possui todos os métodos da classe TClientDataset, permitindo o acesso bidirecional e edição dos registros em memória.

Principais Eventos da Classe TSimpleDataset

Esta classe possui todos os eventos da classe TClientDataset, acrescentando, no entanto, o evento apresentado a seguir.

OnReconcileError

Este evento ocorre sempre que surge um erro na atualização de um registro (numa chamada ao método ApplyUpdates) que não tenha sido tratado pelo evento OnUpdateError.

Capítulo 26

Banco de Dados – Componentes de Acesso via Interbase Express

Neste capítulo serão apresentados os componentes responsáveis pelo acesso a dados via Interbase Express, um mecanismo multiplataforma formado por um conjunto de componentes para acesso nativo ao banco de dados Interbase a partir de uma aplicação desenvolvida com o Delphi 7.

KNOW-HOW em: Componentes de Acesso a Bancos de Dados via Interbase Express

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 7.
- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi 7.
- Compreensão da classe TDataset.

METODOLOGIA

- Apresentação dos componentes de acesso a bancos de dados via Interbase Express, juntamente com uma descrição das suas propriedades, métodos e eventos.

TÉCNICA

- Descrição dos componentes de acesso a bancos de dados via Interbase Express.

Este mecanismo apresenta como principal vantagem o fato de ser um mecanismo de acesso bidirecional, nativo e multiplataforma a bancos de dados do Interbase. Exige, no entanto, o emprego do conceito de transações, que se aplica a aplicações cliente-servidor, apresentadas em maiores detalhes no próximo capítulo.

Estes componentes estão disponíveis na paleta Interbase da paleta de componentes, tanto para aplicações baseadas na VCL como na CLX.

O Componente TIBDatabase

Este componente, derivado por herança da classe TCustomConnection, permite que se estabeleça uma conexão a um banco de dados via Interbase Express. Este componente tem uma funcionalidade semelhante ao TDatabase no mecanismo Borland Database Engine, ao TADOConnection do ADO, e ao TSQLConnection do DBExpress, e permite que diversos componentes de acesso estejam diretamente conectados a ele.

Principais Propriedades da Classe TIBDatabase

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TIBDatabase, além daquelas herdadas das suas classes-base:

AllowStreamedConnection

Essa propriedade é definida como uma variável booleana, e define se uma conexão ao banco de dados deve ser estabelecida a partir do IDE do Delphi, quando a propriedade Connected deste componente for definida como True no Object Inspector.

Connected

Essa propriedade é definida como uma variável booleana, e define se uma conexão foi ou não estabelecida pelo componente.

DatabaseName

Essa propriedade é definida como uma variável do tipo string, e define o nome do banco de dados que será acessado.

DefaultTransaction

Essa propriedade é definida como um objeto da classe TIBTransaction, e define o componente da classe TIBTransaction que representará a transação default para o acesso ao banco de dados.

LoginPrompt

Essa propriedade é definida como uma variável booleana, e define se uma caixa de diálogo de Login deve ou não ser exibida ao se tentar estabelecer uma conexão através deste componente.

Name

Essa propriedade é definida como uma variável do tipo string, e define o nome pelo qual o componente será referenciado no código da aplicação.

Params

Essa propriedade é definida como uma variável do tipo TStrings, e define um conjunto de parâmetros para a conexão que será estabelecida. Como exemplo, podem-se citar as propriedades Username e Password, apresentadas a seguir, que podem ser definidas na caixa de diálogo String List Editor associada a esta propriedade, ao se estabelecer uma conexão quando a propriedade Login Prompt é igual a False.

Password

Essa propriedade é definida como uma variável do tipo string, e define a senha de um usuário que tenta estabelecer a conexão ao banco de dados.

UserName

Essa propriedade é definida como uma variável do tipo string, e define o nome de um usuário que tenta estabelecer a conexão ao banco de dados.

A maioria destas propriedades pode ser configurada usando-se a caixa de diálogo Database Editor, exibida dando-se um duplo clique sobre o componente ou selecionando-se o item correspondente do seu menu pop-up.

SQLDialect

Essa propriedade é definida como uma variável do tipo inteiro, e define o dialeto SQL a ser usado pelas aplicações cliente que acessarão o banco de dados.

Principais Métodos da Classe TIBDatabase

Apresenta-se a seguir uma descrição dos principais métodos da classe TIBDatabase, além daquelas herdadas das suas classes-base.

ApplyUpdates

Declaração

```
procedure ApplyUpdates(const DataSets: array of TDataSet);
```

Esse método aplica atualizações pendentes em objetos derivados da classe TDataset passados como parâmetros, na forma de uma array de objetos da classe TDataset. Desativa todos os objetos derivados da classe TDataset a ele conectados, sem no entanto finalizar a sua própria conexão ao banco de dados.

CloseDataSets

Declaração

```
procedure CloseDataSets;
```

Esse método desativa todos os objetos derivados da classe TDataset a ele conectados, sem no entanto finalizar a sua própria conexão ao banco de dados.

DropDatabase

Declaração

```
procedure DropDatabase;
```

Esse método remove o arquivo associado ao banco de dados, e deve ser usado com muita cautela.

GetTableNames

Declaração

```
procedure GetTableNames(const DatabaseName, Pattern: string; Extensions, SystemTables:
Boolean; List: TStrings);
```

Esse método recebe como parâmetros:

- Um objeto da classe TStrings no qual serão armazenados os nomes das tabelas.
- Uma outra constante booleana que define se os nomes das tabelas de sistema que definem a estrutura das tabelas do banco de dados também devem ser obtidos.

TestConnected

Declaração

```
procedure TestConnected: Boolean;
```

Este método verifica se a conexão ao banco de dados está ativa.

Principais Eventos da Classe TIBDatabase

Apresenta-se a seguir uma descrição dos principais eventos da classe TIBDatabase, além daqueles herdados das suas classes-base.

AfterConnect

Este evento ocorre logo após uma conexão ser estabelecida pelo componente.

AfterDisConnect

Este evento ocorre logo após uma conexão ser finalizada pelo componente.

BeforeConnect

Este evento ocorre imediatamente antes de uma conexão ser estabelecida pelo componente.

BeforeDisConnect

Este evento ocorre imediatamente antes de uma conexão ser finalizada pelo componente.

OnLogin

Este evento ocorre imediatamente após o evento BeforeConnect, e permite que se definam valores para o username e o password de acesso ao banco de dados.

O Componente TIBTransaction

Este componente, derivado por herança da classe TComponent, permite que se represente uma transação individual a um banco de dados representado por um componente TIBDatabase. Um componente TIBDatabase terá uma transação default, mas várias transações podem estar definidas, cada uma definida por um componente Transaction. Os componentes de acesso estarão vinculados a um componente desta classe a partir da sua propriedade Transaction.

O conceito de transações será apresentado no próximo capítulo, que trata de aplicações cliente-servidor.

Principais Propriedades da Classe TIBTransaction

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TIBTransaction, além daquelas herdadas das suas classes-base:

Active

Esta propriedade é definida como uma variável booleana que define se a transação representada pelo componente está ou não ativa.

DatabaseCount

Essa propriedade é definida como uma variável inteira, que define o número de bancos de dados vinculados a essa transação.

Databases

Essa propriedade é definida como uma array de objetos da classe TIBDatabase, e permite acessar diretamente cada um dos componentes que representam os bancos de dados vinculados a essa transação.

DefaultDatabase

Essa propriedade é definida como um objeto da classe TIBDatabase, e permite acessar diretamente o componente que representa o banco de dados default vinculado a essa transação.

InTransaction

Esta propriedade é definida como uma variável booleana que define se a transação representada pelo componente está sendo executada (está em progresso).

Principais Métodos da Classe TIBTransaction

Apresenta-se a seguir uma descrição dos principais métodos da classe TIBTransaction, além daqueles herdados das suas classes-base.

AddDatabase

Declaração

```
function AddDatabase(db: TIBDatabase): Integer;
```

Esse método adiciona um banco de dados, cujo componente TIBDatabase que o representa é passado como parâmetro, à transação representada pelo componente.

CheckDatabasesInList

Declaração

```
procedure CheckDatabasesInList;
```

Esse método verifica se existem bancos de dados associados à transação representada pelo componente, gerando uma exceção caso não haja nenhum.

Commit

Declaração

```
procedure Commit;
```

Esse método confirma a execução de todos os comandos iniciados após a última chamada ao método StartTransaction e finaliza a execução da transação. Esse método só pode ser executado se uma transação estiver sendo processada, o que pode ser constatado verificando-se o valor da propriedade InTransaction. Após uma chamada ao método Commit, a propriedade InTransaction assume o valor False.

CommitRetaining

Declaração

```
procedure CommitRetaining;
```

Esse método confirma a execução de todos os comandos iniciados após a última chamada ao método StartTransaction sem no entanto finalizar a execução da transação. Esse método só pode ser executado se uma transação estiver sendo processada, o que pode ser constatado verificando-se o valor da propriedade InTransaction. Após uma chamada ao método Commit, a propriedade InTransaction assume o valor False.

FindDatabase

Declaração

```
function FindDatabase(db: TIBDatabase): Integer;
```

Esse método retorna o índice de um banco de dados, cujo componente TIBDatabase que o representa é passado como parâmetro, no que se refere à transação representada pelo componente.

FindDatabase

Declaração

```
function FindDefaultDatabase: TIBDatabase;
```

Esse método retorna o componente TIBDatabase que representa o banco de dados default associado à transação.

RemoveDatabase

Declaração

```
procedure RemoveDatabase(Idx: Integer);
```

Esse método remove o banco de dados associado à transação, cujo índice é passado como parâmetro.

RemoveDatabases

Declaração

```
procedure RemoveDatabases;
```

Esse método remove todos os bancos de dados associados à transação.

RollBack

Declaração

```
procedure RollBack;
```

Esse método cancela todos os comandos iniciados após a última chamada ao método StartTransaction e finaliza a execução da transação representada pelo componente. Esse método só pode ser executado se uma transação estiver sendo processada, o que pode ser constatado verificando-se o valor da propriedade InTransaction. Após uma chamada ao método RollBack, a propriedade InTransaction assume o valor False.

RollBackRetaining

Declaração

```
procedure RollBackRetaining;
```

Esse método cancela todos os comandos iniciados após a última chamada ao método StartTransaction sem no entanto finalizar a execução da transação representada pelo componente.

StartTransaction

Declaração

```
procedure StartTransaction;
```

Esse método inicia uma nova transação, e só pode ser executado se não houver uma transação sendo processada, isto é, após uma chamada ao método Commit ou RollBack. Após uma chamada ao método StartTransaction, a propriedade InTransaction assume o valor True.

Principais Eventos da Classe TIBTransaction

Apresenta-se a seguir uma descrição dos principais eventos da classe TIBDatabase, além daqueles herdados das suas classes-base.

OnIdleTimer

Este evento ocorre quando a transação representada pelo componente fica em espera.

A Classe TIBCustomDataset

Conforme descrito anteriormente, a classe TDataSet implementa a funcionalidade genérica para acesso a tabelas, sem incorporar as funções de qualquer API.

A classe TIBCustomDataset, derivada por herança direta da classe TDataSet, incorpora as principais características do mecanismo Interbase Express a alguns dos métodos declarados na classe TDataSet, sobrecarregando-os (no caso de métodos virtuais) ou implementando-os (no caso de métodos abstratos).

Os métodos implementados por essa classe não são normalmente usados no desenvolvimento de aplicações em Delphi, pois geralmente são utilizados componentes representados por classes derivadas por herança desta classe.

Principais Propriedades da Classe TIBCustomDataset

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TDataSet, além daquelas presentes em suas classes ancestrais.

Database

Essa propriedade é definida como uma variável do tipo TIBDatabase, e define o nome do componente Database através do qual será feita a conexão ao servidor do banco de dados.

Active

Esta propriedade é definida como uma variável booleana que define se a conexão representada pelo componente está ou não ativa.

Database

Essa propriedade é definida como uma variável do tipo TIBDatabase, e define o nome do componente IBDatabase através do qual será feita a conexão ao servidor do banco de dados.

DeleteSQL

Essa propriedade é definida como um objeto da classe TStrings, e define o código SQL a ser executado para a exclusão de registros do banco de dados associado através do componente.

InsertSQL

Essa propriedade é definida como um objeto da classe TStrings, e define o código SQL a ser executado para a inclusão de registros do banco de dados associado através do componente.

ModifySQL

Essa propriedade é definida como um objeto da classe TStrings, e define o código SQL a ser executado para a atualização de registros do banco de dados associado através do componente.

RecNo

Essa propriedade é definida como uma variável do tipo inteiro, e define o número do registro corrente dentre aqueles acessados pelo componente.

RecordCount

Essa propriedade é definida como uma variável do tipo inteiro, e retorna o número de registros acessados pelo componente.

RefreshSQL

Essa propriedade é definida como um objeto da classe TStrings, e define o código SQL a ser executado para a atualização da visualização dos registros do banco de dados associado através do componente.

SelectSQL

Essa propriedade é definida como um objeto da classe TStrings, e define o código SQL a ser executado para a consulta a registros do banco de dados associado através do componente.

Transaction

Essa propriedade é definida como um objeto da classe TIBTransaction, e define o componente desta classe ao qual o componente de acesso está vinculado.

Principais Métodos da Classe TIBCustomDataset

Apresenta-se a seguir uma descrição dos principais métodos da classe TIBCustomDataset, além daqueles herdados das suas classes-base.

ApplyUpdates

Declaração

```
procedure ApplyUpdates;
```

Esse método armazena no banco de dados as alterações feitas localmente no componente. Essas alterações, no entanto, só são efetivadas após uma chamada ao método Commit do componente TIBTransaction que representa uma transação do banco de dados.

CancelUpdates

Declaração

```
procedure CancelUpdates;
```

Esse método cancela as alterações feitas localmente para um componente.

Locate

Declaração

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions): Boolean;
```

Esse método permite a busca de um registro por campos que não façam parte do índice corrente da tabela. Recebe como parâmetros:

- Uma string contendo os nomes dos campos pelos quais será feita a pesquisa (separados por ponto-e-vírgula).
- Uma array do tipo Variant contendo os valores a serem pesquisados nos campos pelos quais será feita a pesquisa (separados por ponto-e-vírgula).
- Um conjunto de opções, que pode conter os seguintes elementos:
  - LoCaseInsensitive: Se esse elemento for incluído, letras maiúsculas e minúsculas serão tratadas indiferentemente.
  - LoPartialKey: A pesquisa será aproximada.

Post

Declaração

```
Procedure Post;
```

Esse método efetiva a gravação das alterações no registro corrente acessado pelo componente.

Principais Eventos da Classe TIBCustomDataset

Esta classe implementa os seguintes eventos, além daqueles herdados de sua classe-base:

AfterDatabaseDisconnect

Este evento ocorre logo que a conexão ao servidor do banco de dados é encerrada.

AfterTransactionEnd

Este evento ocorre logo que a transação vinculada ao componente é finalizada.

BeforeDatabaseDisconnect

Este evento ocorre imediatamente antes que a conexão ao servidor do banco de dados seja encerrada.

BeforeTransactionEnd

Este evento ocorre imediatamente antes que a transação vinculada ao componente seja finalizada.

OnUpdateError

Este evento ocorre quando uma exceção é gerada ao se tentar gravar as alterações pendentes em um componente de acesso.

OnUpdateRecord

Este evento ocorre logo ao se atualizar um registro acessado pelo componente.

A Classe TIBDataSet

Esta classe é derivada diretamente da classe TIBCustomDataset, e é implementada na forma de um componente.

A classe TIBDataSet permite que se execute diretamente uma declaração SQL em um banco de dados do Interbase.

Principais Propriedades da Classe TIBDataSet

Apresenta-se a seguir uma descrição das principais propriedades implementadas na classe TIBDataSet, além daquelas herdadas das suas classes ancestrais.

Filtered

Essa propriedade é definida como uma variável booleana, e define se está ou não sendo aplicado um filtro ao conjunto de registros representados pelo componente.

Prepared

Essa propriedade é definida como uma variável booleana, e define se o conjunto de registros representados pelo componente está ou não preparado para a execução de uma consulta parametrizada.

Principais Métodos da Classe TIBDataset

Apresenta-se a seguir uma descrição dos principais métodos da classe TIBDataset, além daqueles herdados das suas classes-base.

ExecSQL

Declaração

```
procedure ExecSQL;
```

Este método executa o comando SQL definido no componente, desde que não corresponda a uma consulta.

ParamByName

Declaração

```
function ParamByName(Idx : String) : TIBXSQLVAR;
```

Este método permite retornar ou definir o valor de um parâmetro em uma consulta parametrizada, sendo o seu nome passado como parâmetro na forma de uma string.

Prepare

Declaração

```
procedure Prepare;
```

Este método prepara o componente para a execução de uma consulta parametrizada pelo componente.

UnPrepare

Declaração

```
procedure UnPrepare;
```

Este método prepara o componente para a execução de uma consulta parametrizada pelo componente.

Principais Eventos da Classe TIBDataset

Esta classe não implementa novos eventos, além daqueles herdados das suas classes-base.

KNOW-HOW em: Classes de Acesso Direto a Bancos de dados via Interbase Express – As Classes TIBTable, TIBQuery e TIBUpdateSQL

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos com o Delphi 7.
- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi 7.
- Compreensão da classes TIBCustomDataset, TIBDataset e TIBDatabase.

METODOLOGIA

- Apresentação das classes e dos componentes de acesso direto a bancos de dados via Interbase Express, juntamente com uma descrição das suas propriedades, métodos e eventos.

TÉCNICA

- Descrição das classes e dos componentes de acesso direto a bancos de dados e apresentação de exemplos de aplicação.
- Serão apresentadas neste tópico as classes usadas para acesso direto a bancos de dados via Interbase Express – as classes TIBTable, TIBQuery e TIBUpdateSQL.

A Classe TIBTable

A classe TIBTable é derivada por herança direta da classe TIBCustomDataSet, sendo normalmente utilizada para se estabelecer uma conexão a uma tabela individual do banco de dados.

Principais Propriedades da Classe TIBTable

Apresenta-se a seguir uma descrição das principais propriedades da classe TIBTable, além daquelas herdadas das suas classes ancestrais.

Active

Esta propriedade é definida como uma variável booleana que define se a conexão representada pelo componente está ou não ativa.

CanModify

Essa propriedade é definida como uma variável booleana e define se a tabela pode ser acessada para edição, inserção e remoção de registros.

Exclusive

Essa propriedade é definida como uma variável booleana e define se a tabela será acessada em modo exclusivo pela aplicação. Se seu valor for igual a True, nenhuma outra aplicação poderá acessar a tabela, enquanto a mesma estiver sendo acessada pela aplicação corrente.

IndexDefs

Essa propriedade é um objeto da classe TindexDefs e define os índices da tabela.

IndexFieldCount

Essa propriedade é definida como uma variável inteira e define o número de campos que compõem o índice corrente.

IndexFieldNames

Essa propriedade é definida como uma variável do tipo string, que armazena os nomes dos campos que compõem o índice corrente, separados por um ponto-e-vírgula.

As propriedades IndexName e IndexFieldNames são mutuamente excludentes, isto é, a definição de um valor para uma propriedade anula o valor definido para a outra.

IndexFields

Essa propriedade é definida como uma array de objetos da classe TField correspondentes aos campos que compõem o índice corrente.

IndexName

Essa propriedade é definida como uma variável do tipo string e define o nome do índice corrente da tabela.

MasterFields

Essa propriedade é definida como uma variável do tipo string e define os nomes dos campos da tabela principal em um relacionamento (separados por um ponto-e-vírgula), devendo ser definida no componente que representa a tabela secundária.

MasterSource

Essa propriedade é definida como um objeto da classe TDataSource e define o nome do componente DataSource ao qual está associado o componente da classe TTable que representa a tabela principal em um relacionamento entre tabelas. Essa propriedade deve ser definida apenas no componente que representa a tabela secundária no relacionamento.

ReadOnly

Essa propriedade é definida como uma variável booleana e define se a tabela pode ser acessada apenas para visualização de registros.

TableName

Essa propriedade é definida como uma variável do tipo string e define os nomes da tabela representada pelo componente.

Principais Métodos da Classe TIBTable

Apresenta-se a seguir uma descrição dos principais métodos da classe TIBTable, além daqueles herdados das suas classes ancestrais.

AddIndex

Declaração

```
procedure AddIndex(const Name, Fields: string; Options: TIndexOptions);
```

Esse método adiciona um índice à tabela, recebendo como parâmetros:

- Uma string com o nome do índice a ser criado.
- Uma string com os nomes dos campos que formam o novo índice (separados por ponto-e-vírgula).
- Uma variável do tipo TIndexOptions, que representa um conjunto que pode incluir os seguintes elementos:
  - ixPrimary: Se o novo índice for definido como a chave primária da tabela (não se aplica a tabelas no formato dBASE).
  - ixUnique: Se o novo índice não admitir duplicidade de valores.
  - ixDescending: Se os registros forem ordenados alfabeticamente de forma decrescente.
  - ixCaseInsensitive: Se não houver diferenciação entre letras maiúsculas e minúsculas na indexação dos campos (não se aplica a tabelas no formato dBASE).
  - ixExpression: O índice será baseado numa expressão (aplica-se apenas a tabelas no formato dBASE).

CreateTable

Declaração

```
procedure CreateTable;
```

Esse método permite a criação de uma tabela em run-time (não confundir com a criação do componente).

Entretanto, antes de se executar uma chamada ao método CreateTable, devem-se definir os valores das seguintes propriedades do componente:

- DataBase: Define o nome do componente TIBDatabase que representa o banco de dados que será acessado.
- TableName: Define o nome da tabela a ser criada.
- TableType: Define o tipo da tabela que será criada.
- FieldDefs: Define os campos da tabela. Conforme será visto posteriormente, esse objeto tem alguns métodos importantes, como:
  - Clear: Remove todas as definições de campos da tabela.
  - Add: Adiciona um novo campo à tabela.

- IndexDefs: Objeto da classe TindexDefs que define os índices da tabela. Essa classe tem alguns métodos importantes, como:
  - Clear: Remove todas as definições de índices da tabela.
  - Add: Adiciona um novo índice à tabela.

DeleteIndex

Declaração

```
procedure DeleteIndex(const Name: string);
```

Esse método remove o índice cujo nome é passado como parâmetro na forma de uma string.

DeleteTable

Declaração

```
procedure DeleteTable;
```

Esse método remove do banco de dados a tabela representada pelo componente, e deve ser empregado com o máximo de cautela.

Antes de se executar esse método, deve-se atribuir o valor False à propriedade Active do componente que representa a tabela (o que também pode ser feito por uma chamada ao seu método Close).

EmptyTable

Declaração

```
procedure EmptyTable;
```

Esse método remove todos os registros da tabela representada pelo componente.

GetIndexNames

Declaração

```
procedure GetIndexNames(List: TStrings);
```

Esse método armazena em um objeto da classe TStrings (passado como parâmetro na chamada do método) os nomes dos índices definidos para a tabela representada pelo componente.

GotoCurrent

Declaração

```
procedure GotoCurrent(Table: TTable);
```

Esse método sincroniza o registro corrente dentre os registros manipulados pelo componente que representa a tabela com o registro corrente de outro componente TTable, cujo nome é passado como parâmetro, e que acessa a mesma tabela.

Principais Eventos da Classe TIBTable

Esta classe não implementa novos eventos, utilizando os herdados das suas classes-base.

A Classe TIBQuery

A classe TIBQuery é derivada por herança direta da classe TIBCustomDataSet, sendo normalmente utilizada para se estabelecer uma conexão a uma ou mais tabelas de um banco de dados, acessadas via Interbase Express usando-se declarações SQL.

Principais Propriedades da Classe TIBQuery

Apresenta-se a seguir uma descrição das principais propriedades da classe TIBQuery, além daquelas herdadas das suas classes ancestrais.

Active

Esta propriedade é definida como uma variável booleana que define se a conexão representada pelo componente está ou não ativa.

Params

Essa propriedade é definida como uma array de objetos da classe TParams, que representam individualmente os parâmetros definidos para a Query.

Prepared

Essa propriedade é definida como uma variável booleana, e define se a Query foi preparada para ser executada, de modo a melhorar o seu desempenho.

A preparação de uma Query pode ser feita atribuindo-se o valor True a essa propriedade, ou mediante uma chamada ao seu método Prepare.

RequestLive

Essa propriedade é definida como uma variável booleana, e define se os registros provenientes de uma consulta podem ser editados localmente pelo usuário.

RowsAffected

Essa propriedade é definida como uma variável inteira, e define o número de linhas ou registros atualizados ou removidos pela execução da última declaração SQL.

SQL

Essa propriedade é definida como um objeto da classe TStrings (que é uma lista de strings) na qual deve ser armazenada a declaração SQL a ser executada mediante uma chamada aos métodos Open ou ExecSQL do componente.

Principais Métodos da Classe TIBQuery

Apresenta-se a seguir uma descrição dos principais métodos da classe TSQLQuery, além daqueles herdados das suas classes ancestrais.

ExecSQL

Declaração

```
Procedure ExecSQL;
```

Esse método permite a execução de declarações SQL que envolvam a inserção, a remoção e a atualização de registros, isto é, declarações SQL que contêm as cláusulas Insert, Delete e Update.

Declarações SQL que envolvem apenas consultas resultantes da utilização da cláusula SELECT devem ser executadas mediante uma chamada ao método Open do componente.

ParamByName

Declaração

```
function ParamByName(Idx : String) : TIBXSQLVAR;
```

Este método permite retornar ou definir o valor de um parâmetro em uma consulta parametrizada, sendo o seu nome passado como parâmetro na forma de uma string.

Prepare

Declaração

```
procedure Prepare;
```

Este método prepara o componente para a execução de uma consulta parametrizada pelo componente.

UnPrepare

Declaração

```
procedure UnPrepare;
```

Este método prepara o componente para a execução de uma consulta parametrizada pelo componente.

Principais Eventos da Classe TIBQuery

Esta classe não implementa novos eventos, utilizando os herdados das suas classes-base.

A Classe TIBUpdateSQL

A classe TIBUpdateSQL é derivada por herança direta da classe TIBDataSetUpdateObject (sendo esta derivada diretamente da classe TComponent), e permite que se definam instruções de inserção (INSERT), deleção (DELETE) e atualização (UPDATE) em registros retornados através de uma consulta SQL, mesmo que esta tenha sido definida como uma consulta apenas de leitura (isto é, a propriedade

RequestLive do componente Query é igual a False ou os registros foram transformados em registros apenas de leitura durante a execução do código), desde que a propriedade CachedUpdates do componente Query responsável pela execução da declaração SQL tenha sido definida como True.

A grande vantagem da utilização desse componente está no fato de não haver necessidade de se preocupar com o fato de os registros terem sido gerados apenas para leitura pelo componente IBQuery.

Principais Propriedades da Classe TIBUpdateSQL

Apresenta-se a seguir uma descrição das principais propriedades da classe TIBUpdateSQL.

DeleteSQL

Essa propriedade é definida como um objeto da classe TStrings, e define o código SQL a ser executado para a exclusão de registros do banco de dados associado através do componente.

InsertSQL

Essa propriedade é definida como um objeto da classe TStrings, e define o código SQL a ser executado para a inclusão de registros do banco de dados associado através do componente.

ModifySQL

Essa propriedade é definida como um objeto da classe TStrings, e define o código SQL a ser executado para a atualização de registros do banco de dados associado através do componente.

RefreshSQL

Essa propriedade é definida como um objeto da classe TStrings, e define o código SQL a ser executado para a atualização da visualização dos registros do banco de dados associado através do componente.

Principais Métodos da Classe TIBUpdateSQL

Apresenta-se a seguir uma descrição dos principais métodos da classe TUpdateSQL.

ExecSQL

Declaração

```
procedure ExecSQL(UpdateKind: TUpdateKind);
```

Esse método executa uma das instruções definidas pelas três propriedades descritas no tópico anterior.

Esse método, ao contrário do método de mesmo nome do objeto Query, recebe como parâmetro uma constante que indica o código a ser executado. A tabela a seguir apresenta essas constantes e seus significados:

- UkModify: Execute a declaração SQL armazenada na propriedade ModifySQL.
- UkInsert: Execute a declaração SQL armazenada na propriedade InsertSQL.
- UkDelete: Execute a declaração SQL armazenada na propriedade DeleteSQL.

Caso as declarações SQL definidas nesse componente tenham parâmetros, os nomes dos parâmetros deverão coincidir com nomes de campos da tabela acessada através do componente Query.

O mecanismo de acesso Interbase Express oferece também o componente IBSQL, da classe TIBSQL, que é uma versão unidirecional, e conseqüentemente mais eficiente para a realização de consultas.

Capítulo 27

Bancos de Dados Cliente/Servidor

Neste capítulo serão apresentados os conceitos fundamentais e indispensáveis à utilização de bancos de dados na filosofia cliente-servidor, utilizando-se como banco de dados o Interbase, desenvolvido e comercializado pela Borland.

KNOW-HOW em: Fundamentos dos Bancos de Dados Cliente/Servidor

PRÉ-REQUISITOS

- Utilização dos componentes de acesso e visualização de bancos de dados do Delphi 7.

METODOLOGIA

- Apresentação do problema: Divisão de tarefas entre uma aplicação cliente e o servidor de bancos de dados.

TÉCNICA

- Apresentação dos procedimentos necessários à criação de um banco de dados no Interbase, à definição de triggers e stored procedures.

Apresentação do Problema

O desenvolvimento de uma aplicação de acesso a banco de dados em ambiente cliente/servidor consiste fundamentalmente em subdividir o trabalho de pesquisa e manutenção de um banco de dados entre a aplicação cliente e o servidor de bancos de dados.

Essa subdivisão de tarefas, no entanto, não deve se limitar apenas a delegar ao servidor a responsabilidade pelo armazenamento de informações. Este deve ser responsável, também, por executar algumas rotinas predefinidas, de modo a limitar os dados a serem enviados ao cliente, reduzindo-se o tráfego de informações na rede.

Conforme será mostrado nos próximos tópicos, o servidor executa essas tarefas mediante a utilização de triggers e stored procedures.

Nos exemplos a serem apresentados, será utilizado o banco de dados Interbase, também desenvolvido pela Borland e que acompanha o Borland Delphi.

O Administrador do Servidor de Banco de Dados

Em um servidor de bancos de dados como o Interbase, cada usuário possui um nome ou identificação (username) e uma senha (password) que devem ser informados sempre que o mesmo quer se conectar ao servidor. Conforme será visto posteriormente, cada usuário possui permissão para realizar um determinado conjunto de ações sobre um banco de dados, e a esse conjunto de ações que podem ser realizadas denomina-se privilégios do usuário.

O Interbase possui um usuário privilegiado, que é o administrador do banco de dados. Esse administrador é inicialmente configurado com o username "SYSDBA" e a senha "masterkey" e, ao acessar o Interbase pela primeira vez após a sua instalação, esses parâmetros deverão ser utilizados.

O administrador do banco de dados pode, entre outras coisas:

- Cadastrar novos usuários.
- Remover usuários cadastrados.
- Alterar as configurações de usuários já cadastrados.

O administrador de um servidor de bancos de dados Interbase realiza essas tarefas através do IBConsole, um utilitário situado no grupo de programas do Interbase e cuja janela principal é mostrada na figura a seguir.

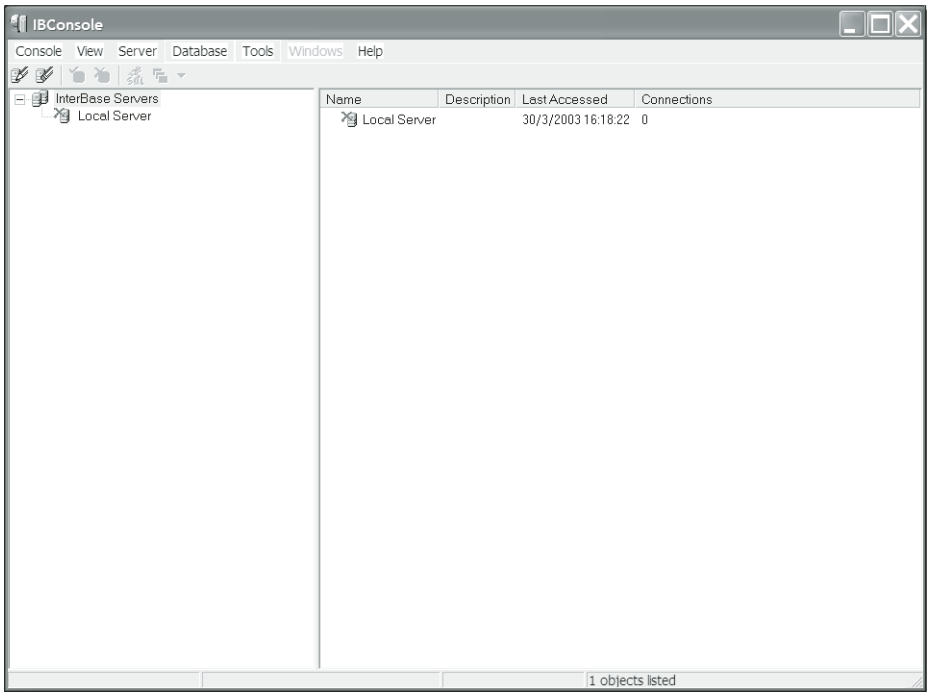


Figura 27.1: O Interbase Server Manager.

Para se conectar ao servidor, o administrador do servidor de bancos de dados deve executar os seguintes procedimentos:

1. Executar o utilitário IBConsole.

2. Selecionar o ícone do servidor de banco de dados a ser conectado e o item Login do menu Server ou dar um duplo clique sobre o ícone selecionado. Será exibida a caixa de diálogo Server Login, na qual o administrador deverá fornecer seu username e sua senha (que inicialmente possuem os valores default descritos anteriormente).

3. Selecionar o botão OK para iniciar a conexão ao servidor. O IBConsole ficará com o aspecto apresentado na figura a seguir.

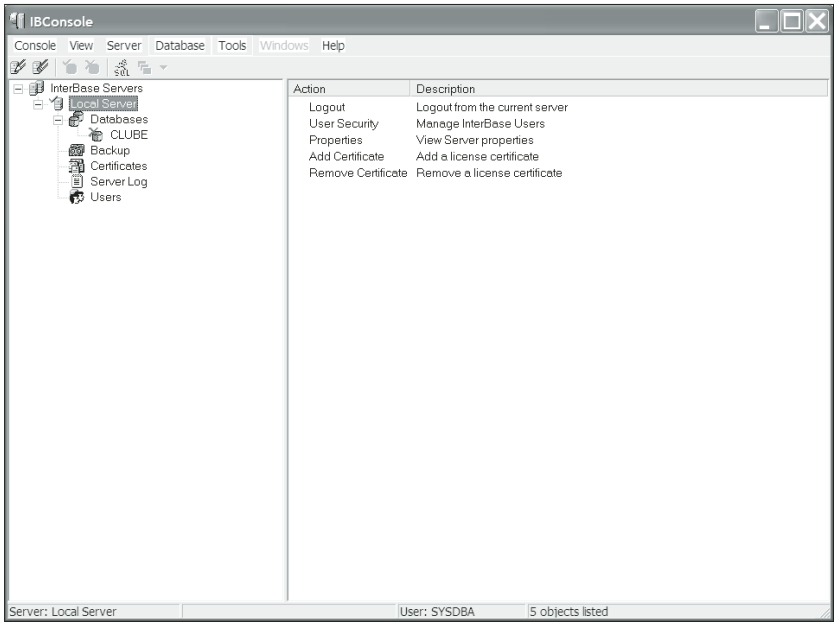


Figura 27.2: A janela do IBConsole, após a conexão inicial.

Para se desconectar do servidor, basta selecionar o servidor de banco de dados e o item Logout do menu Server Manager. Será exibida uma caixa de diálogo na qual será solicitada uma confirmação em relação à desconexão.

Nó próximo tópico, serão apresentados os procedimentos necessários ao cadastramento de um novo usuário.

Cadastrando um Novo Usuário

Após se conectar ao servidor, o administrador pode cadastrar novos usuários, e para isso basta que sejam executados os seguintes procedimentos (após a conexão):

1. Selecionar o ícone correspondente ao servidor e o item User Security do menu Server. Será exibida a caixa de diálogo Interbase security, mostrada na figura a seguir.

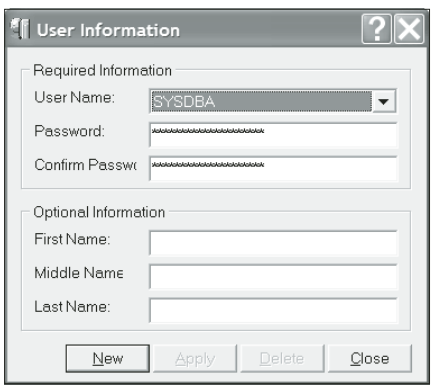


Figura 27.3: A caixa de diálogo User Information.

2. Selecionar o botão New, clicando sobre o mesmo com o botão esquerdo do mouse. Os valores correntes dos campos da caixa de diálogo User Information serão apagados, para que sejam fornecidos os dados do novo usuário, como mostrado na figura a seguir, na qual devem ser fornecidos seu username, sua senha e a confirmação da senha. Opcionalmente, poderá ser fornecido o nome do usuário.

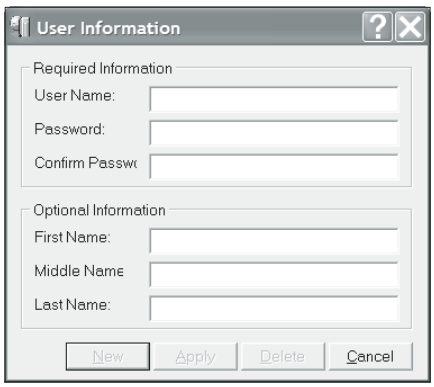


Figura 27.4: A caixa de diálogo User Information.

Alterando os Dados de um Usuário Já Cadastrado

Para alterar os dados de um usuário já cadastrado, você deve acessar o IBConsole Manager e executar os seguintes procedimentos:

1. Selecionar o ícone correspondente ao servidor e o item User Security do menu Server. Será exibida a caixa de diálogo Interbase security, mostrada anteriormente.

2. Selecionar o usuário na relação User name, alterar os dados correspondentes e o botão Apply dessa caixa de diálogo, clicando sobre o mesmo com o botão esquerdo do mouse.

Removendo um Usuário Cadastrado

Para remover um usuário já cadastrado, você deve acessar o IBConsole e executar os seguintes procedimentos:

1. Selecionar o ícone correspondente ao servidor e o item User Security do menu Server. Será exibida a caixa de diálogo Interbase security, mostrada anteriormente.

2. Selecionar o usuário na relação User name e o botão Delete dessa caixa de diálogo, clicando sobre o mesmo com o botão esquerdo do mouse.

3. Selecionar o botão Ok na caixa de diálogo de confirmação que será exibida.

O Utilitário Interactive SQL

Você pode interagir com o servidor empregando o utilitário Interactive SQL.

O utilitário Interactive SQL apresenta a janela principal apresentada na figura a seguir e permite uma interação direta com o Interbase.

Você pode acessar o utilitário Interactive SQL selecionando o item Interactive SQL no menu Tools do IBConsole.

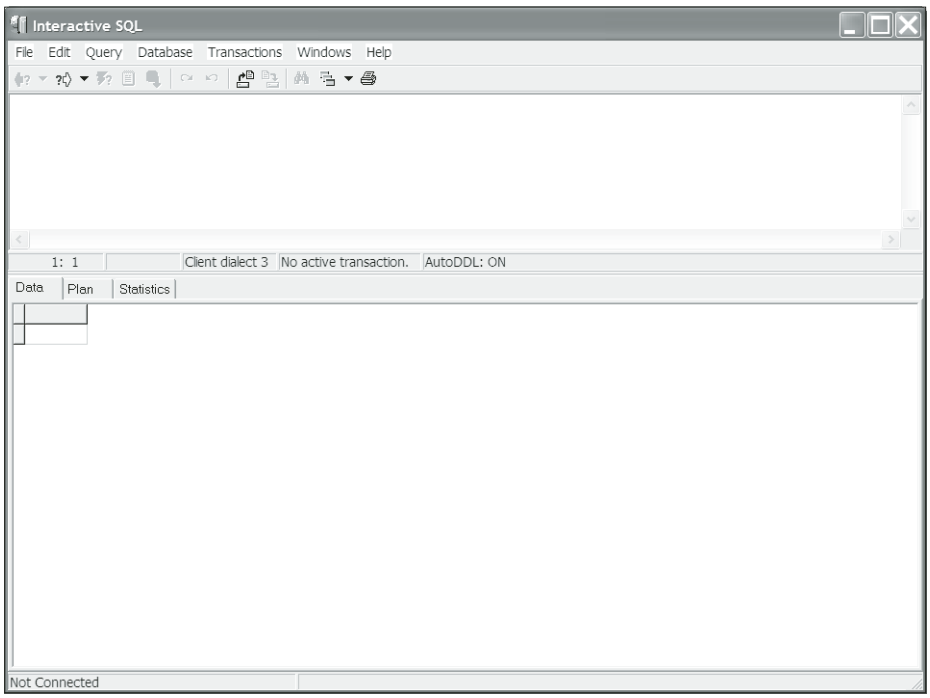


Figura 27.5: A janela principal do utilitário Interactive SQL.

Essa janela é composta por duas áreas distintas:

- Uma área superior (que em versões anteriores era identificada por SQL Statement): É nessa área que devem ser digitadas as declarações SQL que serão utilizadas para a criação de tabelas, inserção, remoção e alteração de registros, etc.
- Uma área inferior (que em versões anteriores era identificada por SQL Output): É nessa área que serão visualizados os resultados de um comando SQL e serão lidas as mensagens emitidas pelo banco de dados, inclusive mensagens de erro.

Criando um Banco de Dados no Interbase a Partir do Utilitário Interactive SQL

Para criar um novo banco de dados a partir do utilitário Interactive SQL, você deve executar os seguintes procedimentos:

1. Selecionar o item Create Database do menu Database. Será exibida a caixa de diálogo Create Database, mostrada na figura a seguir.

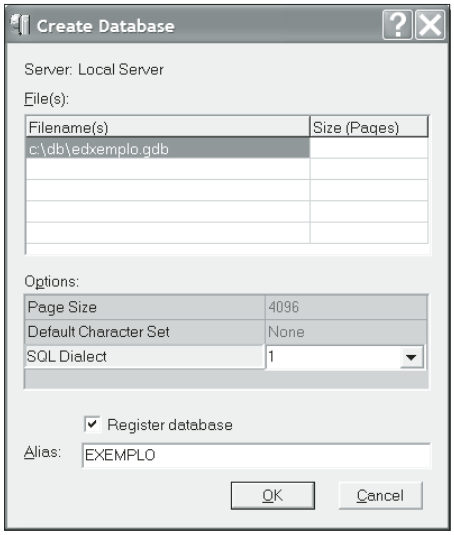


Figura 27.6: A caixa de diálogo Create Database.

2. Digitar, nessa caixa de diálogo, o nome do banco de dados a ser criado (você deve fornecer o path completo e o nome do arquivo com a extensão gdb), como mostrado na figura anterior.

Nesta caixa de diálogo, defina como 1 o valor da propriedade SQL Dialect.

Para se desconectar do banco de dados, você deve executar os seguintes procedimentos:

1. Selecionar o item Disconnect do menu Database. Será exibida uma caixa de diálogo (mostrada na figura a seguir) para que você confirme a ação.

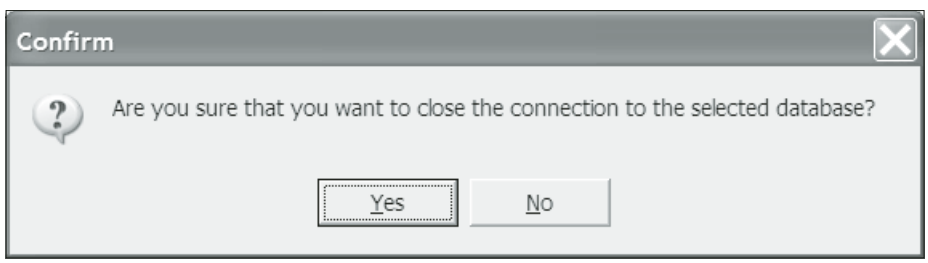


Figura 27.7: Confirmando uma desconexão a um banco de dados do Interbase.

Conectando-se a um Banco de Dados do Interbase

Para se conectar a um banco de dados do Interbase, a partir do utilitário Interactive SQL, você deve executar os seguintes procedimentos:

1. Selecionar o item Connect As do menu Database. Será exibida a caixa de diálogo Database Connect, mostrada na figura a seguir.

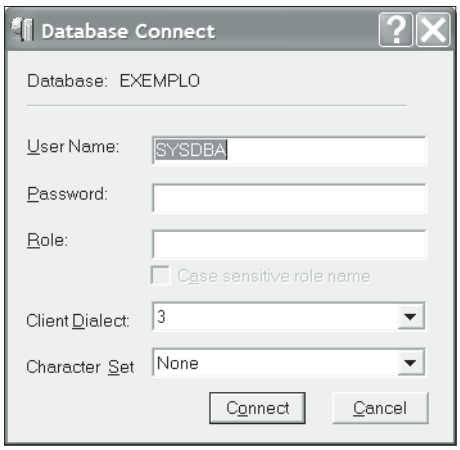


Figura 27.8: A caixa de diálogo Database Connect.

2. Nessa caixa de diálogo, preenchar adequadamente as caixas de texto Username e Password.
3. Selecionar o botão OK, para fechar essa caixa de diálogo e se conectar ao banco de dados.

Tipos de Dados Definidos Pelo Interbase

Os campos de uma tabela do Interbase podem ser de um dos seguintes tipos:

Tipo	Significado	Tamanho
BLOB	Objeto binário	Variável
CHAR(n)	Armazena até *n* caracteres	*n* caracteres (máximo 32767)
DATE	Data/hora	
DECIMAL(p,d)(p = precisão, d = número de casas decimais)		
NUMERIC(p,d)(p = precisão, d = número de casas decimais)		
DOUBLE	Real de precisão dupla	
FLOAT	Real de precisão simples	
INTEGER	Inteiros entre -2,147,483,648 e 2,147,483,648	
SMALLINT	Inteiros entre -32768 e 32767	
VARCHAR(n)	Armazena até *n* caracteres.	

Repare que alguns tipos são bastante semelhantes.

Criando uma Tabela no Interbase

Existem duas formas de se criar uma tabela no Interbase a partir do ambiente Windows:

- Usando o utilitário Interactive SQL.
- Usando o Database Desktop.

Para criar uma tabela do Interbase através do utilitário Interactive SQL, você deve executar os seguintes procedimentos:

1. Conectar-se ao banco de dados, executando os procedimentos descritos nos tópicos anteriores.

2. Digitar um código SQL com a seguinte sintaxe:

```
CREATE TABLE nome da tabela (campo1 tipo1[NOT NULL], … campon tipon[NOT NULL],
```

Na realidade, a sintaxe global desse comando é muito mais complexa, mas vamos nos concentrar numa sintaxe mais simples, de forma a não dificultar a sua compreensão. O parâmetro opcional NOT NULL, se fornecido, indica que esse campo deve ser obrigatório.

Para criar uma tabela chamada Empregados no banco de dados Exemplo.gdb, você deve digitar a seguinte declaração na área superior da janela do utilitário Interactive SQL:

```
CREATE TABLE Empregados (Codigo SMALLINT NOT NULL PRIMARY KEY, Nome VARCHAR(40) NOT NULL,
Funcao VARCHAR(15) NOT NULL, Salario DECIMAL(15,2))
```

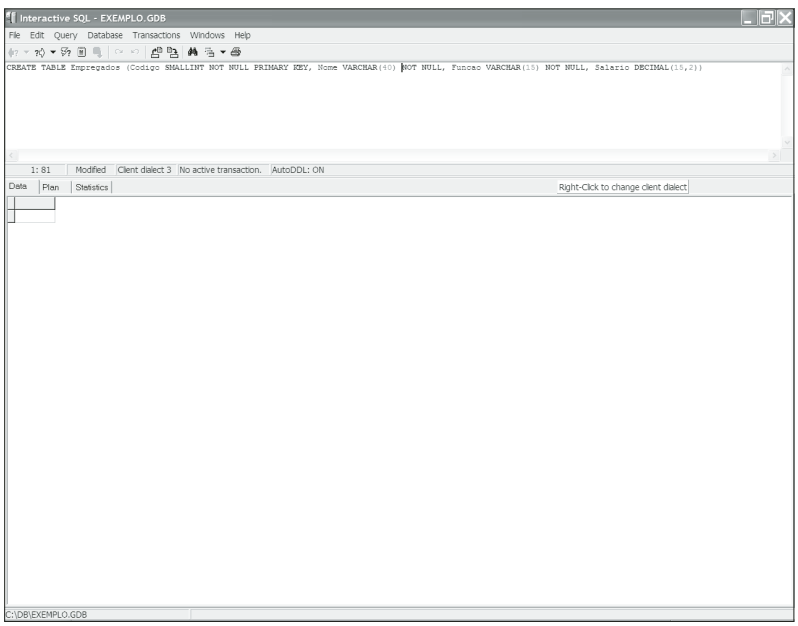


Figura 27.9: Criando uma Tabela no Interbase.

Essa tabela terá a seguinte estrutura:

Nome do Campo	Tipo	NOT NULL	Chave Primária
Codigo	SMALLINT	Sim	Sim
Nome	VARCHAR(40)	Sim	Não
Funcao	VARCHAR(15)	Sim	Não
Salario	DECIMAL(15,2)	Não	Não

3. Selecionar o botão Execute Query (o terceiro botão da bara de ferramentas do utilitário Interactive SQL).

Aplicando Restrições aos Campos de uma Tabela

Durante a criação de uma tabela, os seguintes tipos de restrições podem ser aplicados aos seus campos:

- NULL/NOT NULL: Define se o campo pode ou não ter o valor nulo em algum registro da tabela.
- DEFAULT valor_default: Define um valor default para o campo. Pode ser NULL e, nesse caso, o campo será criado com um valor nulo.
- UNIQUE: Define que dois registros distintos não podem ter o mesmo valor armazenado nesse campo.
- CHECK (condição): Permite que você defina uma condição a ser satisfeita para o valor a ser inserido no campo. Se essa condição não for satisfeita, o valor será rejeitado.
- PRIMARY KEY: Especifica que o campo que está sendo criado será uma chave primária da tabela.
- FOREIGN KEY REFERENCES nome_outra_tabela (campo_de_outra_tabela): Essa restrição indica que o campo que está sendo criado é uma chave primária de outra tabela.

Por exemplo, para garantir que todos os funcionários receberão mais do que R$ 120,00, a declaração SQL que cria a tabela poderia ser redefinida como:

```
CREATE TABLE Empregados (Codigo SMALLINT NOT NULL PRIMARY KEY, Nome VARCHAR(40) NOT NULL,
Funcao VARCHAR(15) NOT NULL, Salario DECIMAL(15,2) CHECK (SALARIO > 120))
```

Removendo uma Tabela do Banco de Dados

Para remover uma tabela de um banco de dados, você deve executar os seguintes procedimentos:

1. Conectar-se ao banco de dados cuja tabela se quer excluir.
2. Executar a seguinte declaração SQL:

```
DROP TABLE nome_da_tabela
```

Criando Índices em uma Tabela

O SQL padrão não define o suporte a índices, mas alguns servidores o suportam, entre os quais o Interbase.

Os índices dão origem a listas ordenadas que permitem uma navegação mais rápida pelos registros de uma tabela, em uma ordenação distinta daquela especificada pelo campo que define a sua chave primária.

A criação de índices no Interbase é feita executando-se uma declaração SQL que apresenta a seguinte sintaxe:

```
CREATE [UNIQUE] [ASC|DESC] INDEX nome_do_indice ON nome_da_tabela (coluna_1,…coluna_n)
```

Para criar um índice chamado IndSalario pelo campo Salário da tabela Empregados, por exemplo, pode-se utilizar a seguinte declaração SQL:

```
CREATE INDEX IndSalario ON Empregados (Salario)
```

Para remover um índice, deve-se executar uma declaração SQL com a seguinte sintaxe:

```
DROP INDEX nome_do_índice
```

Concedendo Privilégios a um Outro Usuário ou Banco de Dados

Suponha que o administrador do servidor de banco de dados tenha criado o banco de dados Exemplo.gdb descrito nos tópicos anteriores, e cadastrado um usuário com username GUESS e uma outra senha qualquer.

Nesse caso, se o usuário GUESS tentar se conectar ao banco de dados Exemplo.gdb, será exibida uma mensagem de erro, pois esse banco de dados não foi criado por ele, e não lhe foi dada qualquer autorização para utilizá-lo.

Quem cria um banco de dados, no entanto, pode conceder esses privilégios a outros usuários (ou a outro banco de dados), conectando-se ao banco de dados e executando um código SQL com a seguinte sintaxe:

```
GRANT privilégio ON [Tabela|Procedure] To [Usuário|Banco de dados]
```

A palavra privilégio deve ser substituída por uma das palavras listadas a seguir:

- Select: Permite uma consulta à tabela (ou View).
- Delete: Permite excluir registros da tabela (ou View).
- Insert: Permite inserir registros na tabela (ou View).
- Update: Permite alterar valores armazenados nos campos dos registros de uma tabela (ou View).
- EXECUTE: Permite a execução de uma stored procedure. Esse privilégio só se aplica no caso de se utilizar a palavra Procedure na cláusula ON.
- ALL: Concede todos os privilégios descritos anteriormente.

Você pode usar a palavra Public para conceder um privilégio a todos os usuários.

Para conceder ao usuário GUESS todos os privilégios de acesso à tabela Empregados, do banco de dados Exemplo.gdb, o administrador deve empregar o seguinte código, após se conectar ao banco de dados:

```
GRANT ALL On Empregados To GUESS
```

Não se esqueça de selecionar o botão Execute Query para executar essa declaração SQL a partir do utilitário Interactive SQL.

Removendo Privilégios

Para remover um privilégio concedido a um usuário, você deve executar uma declaração SQL com a seguinte sintaxe:

```
REVOKE privilégio ON [Tabela|Procedure] FROM [Usuário|Banco de dados]
```

Visões (Views)

Ao acessar um banco de dados usando declarações SQL, muitas vezes executamos repetidas vezes uma instrução SELECT para obter um determinado conjunto de registros que atendem a determinadas condições.

A maioria dos servidores de bancos de dados permite que se crie uma representação alternativa para um conjunto de registros de uma tabela que atendem a determinadas condições, conjunto este denominado View (Visão) e que pode ser tratado como uma tabela independente.

O código de uma declaração SQL utilizado para a criação de uma View apresenta a seguinte sintaxe:

```
CREATE VIEW nome_view (Lista de Colunas)
AS(Código SQL que define a VIEW).
```

A fim de exemplificar a criação de uma View, vamos acessar o banco de dados Employee.gdb, que acompanha o Delphi, executando os procedimentos descritos nos tópicos anteriores. Lembre-se que antes de usar este banco de dados você deve registrá-lo, usando o item Register do menu Database do IBConsole.

Esse banco de dados tem uma tabela chamada Employee, que armazena os dados dos empregados de uma empresa.

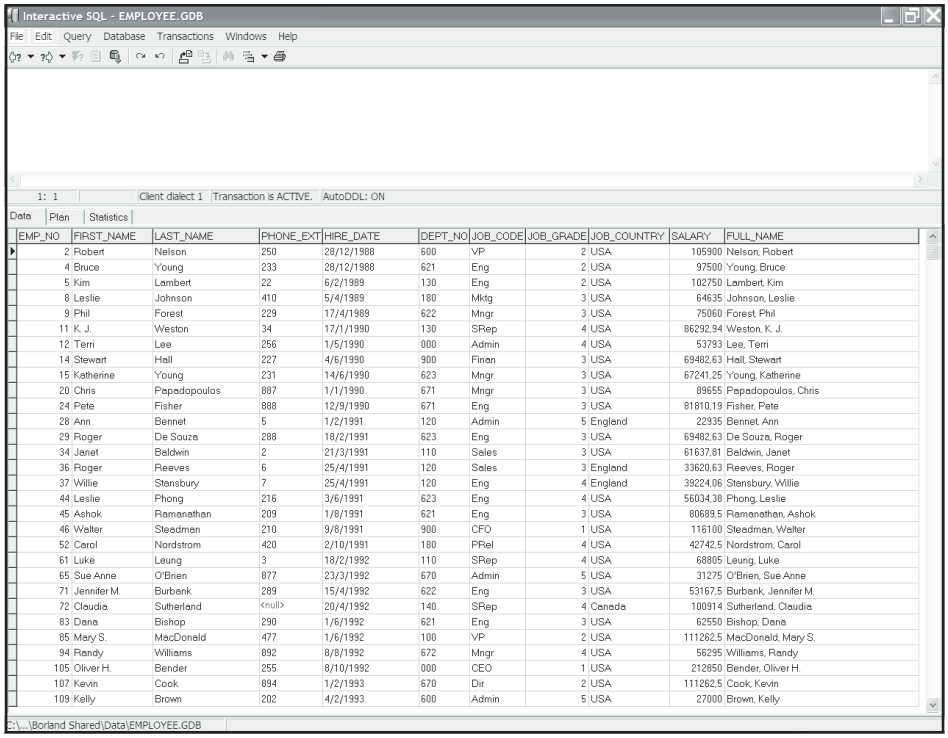
Para criar uma visão dessa tabela chamada Nomes, que retorna apenas o primeiro e o último nomes de cada empregado, deve-se executar a seguinte declaração SQL:

```
CREATE VIEW NOMES (EMP_NO,FIRST_NAME,LAST_NAME) AS
Select * From Employee
```

Você poderá então executar uma consulta SQL sobre a View, como se esta fosse uma tabela independente.

A figura a seguir mostra a execução de uma declaração SQL (SELECT * FROM NOMES) sobre a View recém-criada.

É importante salientar que a Viewé apenas uma representação de um conjunto de registros armazenados em uma tabela. Os registros são fisicamente armazenados na tabela, e não na View.



EMP_NO	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE	DEPT_NO	JOB_CODE	JOB_GRADE	JOB_COUNTRY	SALARY	FULL_NAME
2	Robert	Nelson	250	28/12/1988	600	VP	2	USA	105900	Nelson, Robert
4	Bruce	Young	233	28/12/1988	621	Eng	2	USA	97500	Young, Bruce
5	Kim	Lambert	22	6/2/1989	130	Eng	2	USA	102750	Lambert, Kim
8	Leslie	Johnson	410	5/4/1989	180	Mktg	3	USA	64635	Johnson, Leslie
9	Phil	Forest	229	17/4/1989	622	Mngr	3	USA	75060	Forest, Phil
11	K. J.	Weston	34	17/1/1990	130	SRep	4	USA	86292,94	Weston, K. J.
12	Terri	Lee	256	1/5/1990	000	Admin	4	USA	53793	Lee, Terri
14	Stewart	Hall	227	4/6/1990	900	Finan	3	USA	69482,63	Hall, Stewart
15	Katherine	Young	231	14/6/1990	623	Mngr	3	USA	67241,25	Young, Katherine
20	Chris	Papadopoulos	887	1/1/1990	671	Mngr	3	USA	89655	Papadopoulos, Chris
24	Pete	Fisher	888	12/9/1990	671	Eng	3	USA	81810,19	Fisher, Pete
28	Ann	Bennet	5	1/2/1991	120	Admin	5	England	22935	Bennet, Ann
29	Roger	De Souza	288	18/2/1991	623	Eng	3	USA	69482,63	De Souza, Roger
34	Janet	Baldwin	2	21/3/1991	110	Sales	3	USA	61637,81	Baldwin, Janet
36	Roger	Reeves	6	25/4/1991	120	Sales	3	England	33620,63	Reeves, Roger
37	Willie	Stansbury	7	25/4/1991	120	Eng	4	England	39224,06	Stansbury, Willie
44	Leslie	Phong	216	3/6/1991	623	Eng	4	USA	56034,38	Phong, Leslie
45	Ashok	Ramanathan	209	1/8/1991	621	Eng	3	USA	80689,5	Ramanathan, Ashok
46	Walter	Steadman	210	9/8/1991	900	CFO	1	USA	116100	Steadman, Walter
52	Carol	Nordstrom	420	2/10/1991	180	PRel	4	USA	42742,5	Nordstrom, Carol
61	Luke	Leung	3	18/2/1992	110	SRep	4	USA	68805	Leung, Luke
65	Sue Anne	O'Brien	877	23/3/1992	670	Admin	5	USA	31275	O'Brien, Sue Anne
71	Jennifer M.	Burbank	289	15/4/1992	622	Eng	3	USA	53167,5	Burbank, Jennifer M.
72	Claudia	Sutherland	<null>	20/4/1992	140	SRep	4	Canada	100914	Sutherland, Claudia
83	Dana	Bishop	290	1/6/1992	621	Eng	3	USA	62550	Bishop, Dana
85	Mary S.	MacDonald	477	1/6/1992	100	VP	2	USA	111262,5	MacDonald, Mary S.
94	Randy	Williams	892	8/8/1992	672	Mngr	4	USA	56295	Williams, Randy
105	Oliver H.	Bender	255	8/10/1992	000	CEO	1	USA	212850	Bender, Oliver H.
107	Kevin	Cook	894	1/2/1993	670	Dir	2	USA	111262,5	Cook, Kevin
109	Kelly	Brown	202	4/2/1993	600	Admin	5	USA	27000	Brown, Kelly

Figura 27.10: Resultado da execução de uma declaração SQL sobre a View.

A remoção de uma Viewé feita executando-se uma declaração SQL com a seguinte sintaxe:

```
DROP VIEW nome_da_view
```

O Conceito de Transações

Define-se como transação um conjunto de comandos, na forma de declarações SQL, que devem ser executados antes que as alterações decorrentes desse conjunto de instruções sejam realizadas de forma definitiva no banco de dados.

Uma transação é confirmada mediante a execução do comando COMMIT e cancelada por uma execução do comando ROLLBACK.

A partir do momento em que você inicia uma conexão a um banco de dados, as alterações realizadas (inclusão, remoção e atualização de registros) só serão gravadas de forma efetiva após uma execução do comando COMMIT. Se você executar um ROLLBACK antes de um COMMIT, todas as alterações realizadas desde a execução de último COMMIT serão canceladas.

Incluindo Registros com o Comando INSERT

Para incluir registros em uma tabela de um banco de dados, você deve executar uma declaração SQL com a seguinte sintaxe:

```
INSERT INTO nome_tabela (nome_campo_1,…,nome_campo_n)
     VALUES (valor_campo_1,…,valor_campo_n)
```

O código a seguir insere um registro na tabela Empregados, criada anteriormente:

```
INSERT INTO Empregados (Codigo,Nome,Funcao,Salario) VALUES (101,'Marcelo
Leão','Professor',2000.00)
```

Neste exemplo foi usado o dialeto 3 da linguagem SQL.

Executando os procedimentos descritos anteriormente, inclua os seguintes registros na tabela Empregados, do banco de dados Exemplo.gdb:

Código	Nome	Função	Salário
101	Marcelo Leão	Professor	2000.00
102	José Vieira	Diretor	2500.00
103	Maria Emilia	Secretária	1200.00
104	Rosane Santos	Bibliotecária	1800.00

ATUALIZANDO REGISTROS EM UMA TABELA

Para atualizar registros em uma tabela de um banco de dados, você deve executar uma declaração SQL com a seguinte sintaxe:

```
UPDATE nome_tabela SET nome_campo_1=Valor_1,…,nome_campo_n=Valor_n
WHERE (Condição)
```

Por exemplo, para atualizar os salários de maneira que todos os empregados recebam, no mínimo, R$ 1.400,00, pode-se executar uma declaração SQL como a apresentada a seguir.

```
UPDATE Empregados SET Salario = 1400.00 Where (Salario < 1400)
```

Se você está executando esses procedimentos no seu computador, execute um COMMIT para gravar as alterações feitas até o momento.

REMOVENDO REGISTROS DE UMA TABELA

Para remover registros de uma tabela de um banco de dados, você deve executar uma declaração SQL com a seguinte sintaxe:

```
DELETE FROM nome_tabela WHERE (Condição)
```

Por exemplo, para remover os empregados com salário superior a R$ 2000,00, você deve executar a seguinte declaração SQL:

```
DELETE FROM Empregados WHERE (Salario > 2000.00)
```

Teste esse comando, visualize os registros da tabela com uma instrução SELECT e execute um ROLLBACK para recuperar esse registro.

Ordenando os Registros de uma Tabela

Para relacionar os registros de uma tabela ordenados por um determinado campo (ou por um grupo de campos), você deve executar uma declaração SELECT com uma cláusula ORDER BY, que apresenta a seguinte sintaxe:

```
SELECT (Lista de Colunas) FROM nome_da_tabela WHERE (Condição) ORDER BY nome_coluna_1
[ASC|DESC],…, nome_coluna_n [ASC|DESC]
```

Nesse código, ASC e DESC significam ordem Ascendente (Crescente) e Descendente (Decrescente), respectivamente, onde a primeira opção é a opção default.

Por exemplo, para ordenar os empregados em ordem crescente de salário, basta executar a seguinte declaração SQL:

```
Select * From Empregados ORDER BY Salario
```

TRIGGERS

As triggers são um conjunto de funções definidas em um banco de dados e que podem ser executadas em decorrência de instruções SQL que envolvem a inclusão de novos dados (INSERT), atualização (UPDATE) ou remoção de dados já existentes (DELETE).

Pode-se então fazer uma analogia entre as triggers e os procedimentos associados a eventos do Delphi e dizer que "as triggers são procedimentos associados a eventos do banco de dados", e esses eventos podem ser de seis tipos distintos:

- BEFORE INSERT
- AFTER INSERT
- BEFORE UPDATE
- AFTER UPDATE
- BEFORE DELETE
- AFTER DELETE

A princípio, pode-se definir mais de uma trigger para cada evento no Interbase, razão pela qual cada trigger receberá um número (entre 0 e 32767) que a identificará unicamente entre as demais triggers associadas a um mesmo evento. Esse número definirá a ordem em que as triggers serão executadas, e será definida pela cláusula POSITION na declaração SQL usada para criar a TRIGGER.

É recomendável que as triggers não sejam numeradas seqüencialmente, mas em incrementos que possibilitem a inserção de outras triggers.

O código utilizado para a criação de uma trigger no Interbase apresenta a seguinte sintaxe:

```
SET TERM ^ ;
CREATE TRIGGER nome_trigger FOR nome_tabela evento POSITION número AS
BEGIN
     // Código da Trigger
END ^
```

onde a palavra "evento" deverá ser substituída por um dos seis eventos descritos anteriormente.

Por exemplo, para evitar que um funcionário seja cadastrado com um salário superior a R$ 5000,00, você pode criar as seguintes triggers para a tabela Empregados, do banco de dados Exemplo.gdb:

```
SET TERM ^ ;
CREATE TRIGGER MAXSALARIO FOR Empregados AFTER INSERT POSITION 5 AS
BEGIN
     UPDATE Empregados SET Salario = 5000.00 WHERE (Salario > 5000.00);
END ^

CREATE TRIGGER LIMITASALARIO FOR Empregados AFTER UPDATE POSITION 5 AS
BEGIN
     UPDATE Empregados SET Salario = 5000.00 WHERE (Salario > 5000.00);
END ^
```

Se você tentar inserir um funcionário com salário superior a R$ 5000,00 verá que esse limite será respeitado, isto é, o valor 5000,00 será armazenado no campo Salario.

A remoção de uma trigger é feita em uma instrução SQL com a seguinte sintaxe:

```
DROP TRIGGER nome_trigger
```

Para alterar a definição de uma TRIGGER, você deve executar uma instrução SQL com a seguinte sintaxe:

```
SET TERM ^ ;
ALTER TRIGGER nome_trigger evento POSITION número AS
BEGIN
     // Código da Trigger
END ^
```

A instrução SET TERM define o símbolo usado para a terminação da TRIGGER, que deve ser diferente do ponto-e-vírgula, usado na linguagem de codificação do Interbase. O mesmo será válido para stored procedures.

STORED PROCEDURES (PROCEDIMENTOS ARMAZENADOS)

As stored procedures (também conhecidas como "procedimentos armazenados") são funções armazenadas no banco de dados, e que podem ser executadas a partir de uma instrução SQL (EXECUTE PROCEDURE), de outra stored procedure, de uma trigger ou de uma aplicação-cliente desenvolvida em Delphi.

O código utilizado para criar uma stored procedure no Interbase apresenta a seguinte sintaxe:

```
SET TERM ^  ;
CREATE PROCEDURE nome_procedure (Lista de Parâmetros de Entrada)
RETURNS (Lista de Parâmetros de Saída) AS
```

```
BEGIN
     // Código da Stored Procedure
END ^
```

As listas de parâmetros de entrada e de saída devem especificar os nomes dos parâmetros e seus tipos.

A remoção de uma stored procedure é feita em uma instrução SQL com a seguinte sintaxe:

```
DROP PROCEDURE nome_stored_procedure
```

A stored procedure apresentada a seguir permite que se multiplique por um fator o salário de todos os empregados.

```
SET TERM ^ ;
CREATE PROCEDURE AUMENTO (Fator Float) AS
BEGIN
     UPDATE Empregados SET Salario = Salario * :Fator;
END ^
```

Repare que, no código da stored procedure, o parâmetro fator é precedido por dois-pontos (:).

Para executar uma stored procedure, pode-se utilizar a seguinte declaração SQL:

```
EXECUTE PROCEDURE nome_stored_procedure(parâmetros de entrada)
     RETURNING_VALUES (Lista de parâmetros de saída)
```

Por exemplo, para aumentar em 10% o salário de todos os empregados cujos dados estão armazenados nos registros da tabela Empregados, basta que se execute a seguinte declaração SQL:

```
EXECUTE PROCEDURE AUMENTO(1.10)
```

Para alterar a definição de uma Stored Procedure, você deve executar uma instrução SQL com a seguinte sintaxe:

```
SET TERM ^ ;
ALTER PROCEDURE nome_procedure (Lista de Parâmetros de Entrada)
RETURNS (Lista de Parâmetros de Saída) AS
BEGIN
     // Código da Stored Procedure
END ^
```

Linguagem de Codificação do Interbase

O Interbase tem uma linguagem de codificação própria, que pode ser usada na definição de triggers e stored procedures mais complexas.

Declaração de Variáveis no Interbase

A declaração de variáveis internas a uma trigger ou stored procedure deve ser feita antes da palavra reservada BEGIN (que define o início do corpo principal da trigger ou stored procedure) e apresenta a seguinte sintaxe:

```
DECLARE VARIABLE nome_variável tipo_variável;
```

Deve-se utilizar uma instrução DECLARE VARIABLE para cada variável a ser declarada.

Atribuição de Valores a Variáveis no Interbase

A atribuição de um valor a uma variável interna de uma stored procedure é feita em uma linha de código que apresenta a seguinte sintaxe:

```
Nome_variável = valor;
```

Definição de Comentários no Interbase

As triggers e stored procedures definidas no Interbase podem conter comentários semelhantes aos utilizados na linguagem C, como mostra a linha de código a seguir.

```
/* Isto é um Comentário */
```

Estruturas Condicionais do Interbase

O Interbase tem uma estrutura condicional cuja sintaxe é apresentada a seguir.

```
If (condição)
     Then
             // Comandos executados se a condição for verdadeira;
     Else
             // Comandos executados se a condição for falsa;
```

Diferentemente do que ocorre no Pascal, no Interbase deve-se colocar um ponto-e-vírgula antes da cláusula Else de uma estrutura condicional.

A procedure AUMENTO, por exemplo, poderia ser reescrita da seguinte maneira:

```
CREATE PROCEDURE AUMENTO (Fator Float) AS
BEGIN
     If (:Fator > 1.0) Then
                 UPDATE Empregados SET Salario = Salario * :Fator;
END
```

Estruturas de Repetição do Interbase

O Interbase apresenta as estruturas de repetição FOR SELECT…DO SUSPEND e WHILE…DO.

As estruturas de repetição FOR SELECT…DO SUSPEND apresentam a seguinte sintaxe:

```
FOR
             SELECT (restante da declaração Select)
             INTO (Lista de parâmetros)
     DO SUSPEND;
```

Para a tabela Empregados, do banco de dados Exemplo.gdb, pode-se criar a seguinte procedure, que retorna o nome e o salário de cada empregado:

```
CREATE PROCEDURE DADOS
     RETURNS (NOME VARCHAR(40), SALARIO DECIMAL(15,2)) AS
BEGIN
```

```
FOR
       SELECT Nome,Salario From Empregados
       INTO:Nome, :Salario
DO SUSPEND;
END
```

Nas estruturas de condição e de repetição descritas anteriormente, podem-se utilizar blocos de comando BEGIN...END, como no Object Pascal.

Criação de Novos Tipos no Interbase

A linguagem SQL permite que você crie um tipo de dado (chamado Domínio) baseado em um tipo já definido.

A criação de um domínio é feita mediante a execução de uma declaração SQL que apresenta a seguinte sintaxe:

```
CREATE DOMAIN nome_do_domínio AS tipo_de_dado DEFAULT valor_default
```

A especificação de um valor default para um domínio é opcional.

Consideremos novamente a tabela Empregados, em que o campo Salario foi definido como DECIMAL(15,2).

Nesse caso, se utilizarmos muitas tabelas com esse tipo de campo, podemos criar um domínio chamado Salario, mediante a execução de uma instrução SQL como a apresentada a seguir.

```
CREATE DOMAIN SALARIO AS DECIMAL(15,2)
```

A partir daí, SALARIO poderia ser tratado como um tipo da linguagem, e poderíamos criar variáveis desse novo tipo.

Para remover um domínio, você deve executar uma instrução SQL com a seguinte sintaxe:

```
DROP DOMAIN nome_do_domínio
```

Você só pode remover um domínio se ele não estiver sendo usado como definição de campos em uma das tabelas definidas no banco de dados.

Para alterar a definição de um domínio, deve-se executar uma instrução SQL com a seguinte sintaxe:

```
ALTER DOMAIN nome_domínio AS Tipo_Prédefinido
```

Metadados de um Banco de Dados

Quando um banco de dados é criado no Interbase, ele armazena dois tipos de informações: o primeiro tipo, denominado Metadados, diz respeito às definições das tabelas, índices, triggers, stored procedures e outros itens que definem a estrutura do banco de dados; o segundo tipo se refere aos dados realmente armazenados nos registros das tabelas que compõem o banco de dados.

Para visualizar os metadados de um banco de dados do Interbase, você deve executar os seguintes procedimentos:

1. Conectar-se ao banco de dados, dando um duplo clique sobre o mesmo no IBConsole.
2. Selecionar o item View Metadata do menu Database do IBConsole.

Os metadados do banco de dados serão exibidos na janela Database Metadata.

Para visualizar os metadados de uma única tabela, você deve executar os seguintes procedimentos:

1. Conectar-se ao banco de dados, dando um duplo clique sobre o mesmo no IBConsole.
2. Selecione o item Tables entre as opções disponíveis para o banco de dados a partir do IBConsole. As tabelas do banco de dados serão exibidas no painel direito do IBConsole.
3. Dê um duplo clique sobre a tabela desejada, no painel direito do IBConsole.
4. Selecione a guia Metadata na janela que será exibida, e a partir da qual você poderá selecionar outras tabelas cujos metadados desejar visualizar.

Você também pode visualizar os metadados de uma View, desde que existam Views definidas no banco de dados.

Concatenando Dados Provenientes de Várias Tabelas

Você pode utilizar instruções SELECT complexas, que retornem dados provenientes de várias tabelas e que atendam a determinadas condições.

Por exemplo, para concatenar os registros das tabelas Employee e Depart, do banco de dados Employee.gdb, reunidas pelo campo que define o código do departamento em cada tabela, pode-se usar uma declaração SQL como a exemplificada a seguir.

```
SELECT * FROM employee, department where (employee.dept_no = department.dept_no)
```

Ao executar essa declaração SQL, os campos dos registros da tabela Employee precedem os campos da tabela Department.

Se a declaração SQL for definida como a exemplificada a seguir, a ordem de exibição dos campos será invertida.

```
SELECT * FROM department,employee where (employee.dept_no = department.dept_no)
```

Para restringir os campos a serem exibidos, o nome de cada campo deve ser exibido na declaração, como exemplificado a seguir.

```
SELECT employee.First_Name,employee.Last_Name,Department.Department FROM employee,department
where (employee.dept_no = department.dept_no)
```

Nesse caso, apenas os campos selecionados serão exibidos.

Caso o número de campos a serem exibidos seja muito grande, o fato de ter de preceder o nome do campo pelo nome da tabela pode ser uma tarefa bastante ingrata. Nesse caso, pode-se usar um nome alternativo para a tabela (que deve ser definido na própria declaração SELECT), como mostra o exemplo de instrução SQL mostrado a seguir:

```
SELECT e.First_Name,e.Last_Name,d.Department FROM employee e,department d where (e.dept_no =
d.dept_no)
```

Nesse caso, "e" se refere à tabela employee e "d" se refere à tabela department.

Criando um Backup de um Banco de Dados do Interbase

Para fazer um backup de um banco de dados criado no Interbase, você deve executar os seguintes procedimentos:

1. Executar o utilitário IBConsole.

2. Conectar-se ao servidor de banco de dados, executando os procedimentos descritos nos tópicos anteriores.

3. Selecionar o banco de dados desejado e, no menu Database do IBConsole, selecionar o item Maintenance -> Backup/Restore -> Backup. Será exibida a caixa de diálogo Database Backup, na qual deve ser selecionado o nome do banco de dados do qual se deseja fazer o backup e o nome do arquivo no qual o backup será feito.

4. Selecionar o botão OK, para fechar essa caixa de diálogo e realizar o backup.

Será exibido um relatório do backup, e uma caixa de diálogo informando a sua conclusão.

Recuperando um Banco de Dados a Partir de um Backup

Para recuperar um backup de uma base de dados, você deve executar os seguintes procedimentos:

1. Executar o utilitário IBConsole.

2. Conectar-se ao servidor de banco de dados, executando os procedimentos descritos nos tópicos anteriores.

3. Selecionar o banco de dados desejado e, no menu Database do IBConsole, selecionar o item Maintenance -> Backup/Restore -> Restore. Será exibida a caixa de diálogo Database Restore, na qual deve ser selecionado o nome do arquivo a partir do qual será feita a restauração do banco de dados, e o arquvo de destino (que representa o banco de dados que será restaurado).

4. Selecionar o botão OK, para fechar essa caixa de diálogo e realizar o backup.

Criando um Alias Para um Banco de Dados do Interbase

Para criar um alias para um banco de dados do Interbase, você deve executar os seguintes procedimentos:

1. Selecionar o item Explorer do menu Database do Delphi 7. Será exibida a caixa de diálogo SQL Explorer.

2. Selecionar o item New do menu Object. Será exibida a caixa de diálogo New Database Alias, na qual deverá ser selecionada a opção correspondente ao Interbase, e o botão OK para fechar essa caixa de diálogo. O novo alias será criado com um nome default (INTRBASE1, no caso deste exemplo).

3. Alterar o nome do alias para outro nome mais fácil de memorizar (neste exemplo, usou-se o nome AXCEL).

4. Definir o nome e o path do arquivo que armazena o banco de dados na opção SERVER NAME, como mostrado na figura a seguir.

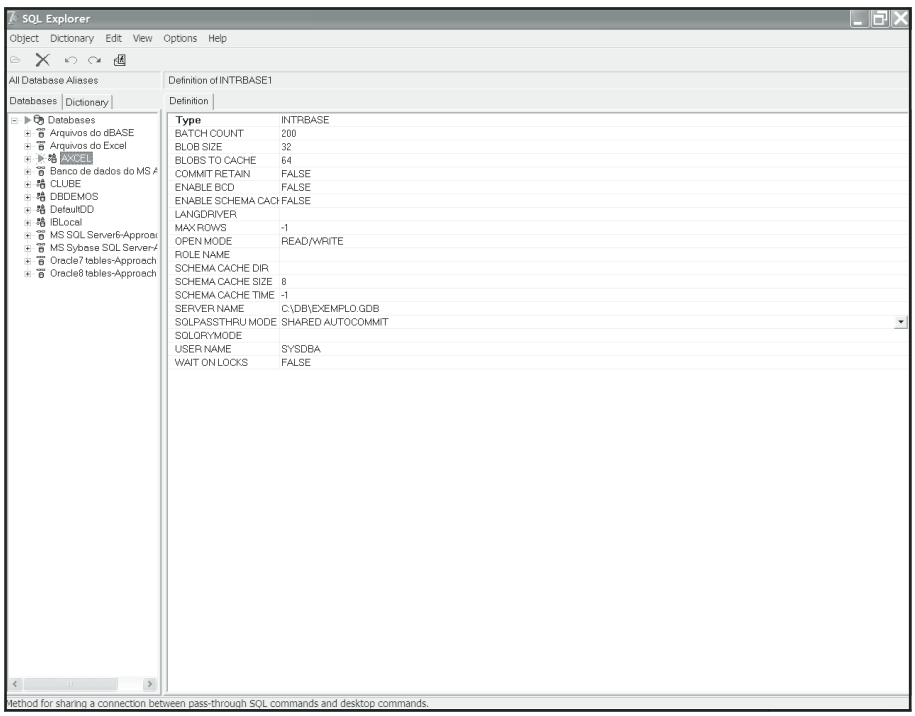


Figura 27.11: Criando um alias para um banco de dados do Interbase.

5. Definir o nome do usuário na opção USER NAME.

6. Fechar a janela do SQL Explorer salvando as alterações feitas no alias, selecionando o item Apply do menu Object ou o botão Apply da barra de ferramentas do SQL Explorer.

Pronto! Seu alias foi criado e pode ser selecionado na lista de opções disponíveis na propriedade DatabaseName dos componentes Table e Query.

Capítulo 28

Acessando Bancos de Dados Cliente/Servidor

Neste capítulo serão apresentados os procedimentos necessários ao acesso a bancos de dados cliente-servidor a partir de uma aplicação desenvolvida em Delphi, usando os diversos mecanismos de acesso, e principalmente os componentes que permitem a execução de stored procedures (procedimentos armazenados) a partir de uma aplicação.

KNOW-HOW em: Acesso a Bancos de Dados Cliente/Servidor

PRÉ-REQUISITOS

- Utilização dos componentes de acesso e visualização de bancos de dados do Delphi 7.

METODOLOGIA

- Apresentação do problema: Divisão de tarefas entre uma aplicação cliente e o servidor de bancos de dados.

TÉCNICA

- Apresentação dos procedimentos necessários ao acesso a bancos de dados cliente/servidor usando os diversos mecanismos de acesso.

Apresentação do Problema

O desenvolvimento de uma aplicação de acesso a bancos de dados em ambiente cliente/servidor consiste fundamentalmente em subdividir o trabalho de pesquisa e manutenção de um banco de dados entre a aplicação cliente e o servidor de bancos de dados.

Essa subdivisão de tarefas, no entanto, não deve se limitar apenas a delegar ao servidor a responsabilidade pelo armazenamento de informações. Este deve ser responsável, também, por executar algumas rotinas predefinidas, de modo a limitar os dados a serem enviados ao cliente, reduzindo-se o tráfego de informações na rede.

Conforme mostrado no capítulo anterior, o servidor executa essas tarefas mediante a utilização de triggers e stored procedures.

Acessando Banco de Dados Cliente-Servidor do Interbase em uma Aplicação Delphi via BDE

A princípio, o acesso a um banco de dados cliente-servidor criado no Interbase não oferece maiores dificuldades.

No caso de se usar o acesso via BDE, você pode inserir, em um formulário (ou em um Datamodule), componentes Table ou Query, definir a propriedade Databasename desses componentes como o alias criado para o banco de dados, definir adequadamente a propriedade TableName (no caso do componente Table) ou SQL (no caso do componente Query) e definir como True a propriedade Active do componente empregado.

A principal diferença é que, nesses casos, será exibida uma caixa de diálogo para Login, como mostra a figura a seguir.

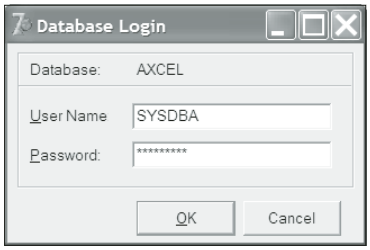


Figura 28.1: A caixa de diálogo Database Login.

Em seguida, basta ligar um DataSource ao componente Table ou Query (através da sua propriedade DataSet), inserir diversos componentes de visualização no formulário (configurando adequadamente suas propriedades DataSource e DataField) e, se for o caso, um componente DBNavigator.

Ao executar esses procedimentos, no entanto, não estaremos aproveitando muitas das vantagens de se empregar uma ferramenta de desenvolvimento cliente/servidor.

Conforme será mostrado a seguir, o Delphi tem alguns componentes que nos ajudam a explorar as potencialidades dessa filosofia.

Por exemplo, para executar uma stored procedure a partir de uma aplicação desenvolvida em Delphi através de mecanismo de acesso do Borland Database Engine, você deve utilizar o componente StoredProc, existente na página BDE da paleta de componentes.

Apresentam-se a seguir as principais propriedades, métodos e eventos desse componente, derivado por herança da classe TDBDataset.

Principais Propriedades do Componente StoredProc

Apresentam-se a seguir as principais propriedades desse componente, além de outras herdadas das suas classes ancestrais.

Active

Essa propriedade é definida como uma variável booleana, e pode ser definida como True se o procedimento armazenado se baseia numa simples consulta.

Database

Essa propriedade é definida como um objeto da classe TDatabase, e permite o acesso ao componente que representa o banco de dados ao qual se está conectado.

É importante lembrar que, se nenhum componente TDataBase for explicitamente incluído na aplicação, uma instância dessa classe será gerada automaticamente pelo Delphi.

DatabaseName

Essa propriedade é definida como uma variável do tipo string, e define o nome do banco de dados ao qual o componente está conectado. Pode ser o nome de um alias definido no Borland Database Engine ou o nome definido na propriedade DataBaseName do componente Database associado.

ParamCount

Essa propriedade é definida como uma variável do tipo Word, e define o número de parâmetros da stored procedure que será executada pelo componente, parâmetros estes que podem ser visualizados através da propriedade Params do componente.

Params

Essa propriedade é definida como uma array de objetos da classe TParams, que representam individualmente os parâmetros definidos para a stored procedure. Para acessar a caixa de diálogo na qual são definidos esses parâmetros, basta selecionar, com o botão esquerdo do mouse, as reticências exibidas à direita do nome da propriedade no Object Inspector.

Prepared

Essa propriedade é definida como uma variável booleana, e define se a stored procedure foi preparada para ser executada, de modo a melhorar a sua performance.

A preparação de uma stored procedure pode ser feita atribuindo-se o valor True a essa propriedade, ou mediante uma chamada ao seu método Prepare.

StoredProcName

Essa propriedade é definida como uma variável do tipo string, e define o nome da stored procedure a ser executada.

As stored procedures disponíveis para o banco de dados podem ser selecionadas na lista drop-down exibida à direita do nome da propriedade.

Principais Métodos do Componente StoredProc

Apresentam-se a seguir os principais métodos desse componente, além de outras herdadas das suas classes ancestrais.

CopyParams

Declaração

```
procedure CopyParams(Value: TParams);
```

Esse método copia a lista de parâmetros do componente para outra lista de parâmetros (que pode ser a propriedade Params de outro componente StoredProc).

ExecProc

Declaração

```
procedure ExecProc;
```

Esse procedimento executa no servidor a procedure associada ao componente, e cujo nome é definido na sua propriedade StoredProcName.

ParamByName

Declaração

```
function ParamByName(const Value: string): TParam;
```

Esse método permite o acesso individual a parâmetros definidos em uma stored procedure, sendo o nome de um parâmetro passado na forma de uma string na chamada ao procedimento.

Prepare

Declaração

```
procedure Prepare;
```

Esse método prepara a stored procedure a ser executada, atribuindo o valor True à sua propriedade Prepared.

UnPrepare

Declaração

```
procedure UnPrepare;
```

Esse método cancela a preparação da stored procedure a ser executada, atribuindo o valor False à sua propriedade Prepared.

Principais Eventos do Componente IBoredProc

Apresentam-se a seguir os principais eventos deste componente, além daqueles herdados das suas classes ancestrais.

OnUpdateError

Esse evento ocorre quando um erro é gerado ao se tentar atualizar alterações pendentes no banco de dados.

OnUpdateRecord

Esse evento ocorre ao se atualizar alterações pendentes no registro corrente.

Exemplo de Aplicação

Para criar o aplicativo-exemplo, você deve executar os seguintes procedimentos:

1. Selecionar o item File/New Application no menu do Delphi, para iniciar uma nova aplicação baseada na VCL.

2. Insira um componente Datasource no formulário e defina sua propriedade Name como sendo igual a "DatasourceCS".
3. Insira neste formulário um componente DBGrid e defina sua propriedade Name como DBGridCS.
4. Insira neste formulário um componente DBNavigator e defina sua propriedade Name como DBNavigarorCS.
5. Defina a propriedade Datasource destes componentes como DatasourceCS.
6. Insira neste formulário um componente Label e defina sua propriedade Caption como 'Valor'.
7. Insira neste formulário um componente Edit e defina sua propriedade Name como EditValor. Defina como '1.0' o texto exibido na propriedade Text deste componente.
8. Altere a propriedade Caption deste formulário para 'Exemplo de Acesso a Banco de Dados Cliente-Servidor'.
9. Insira um componente BitBtn neste formulário e atribua os seguintes valores para as suas principais propriedades:

   Name: BotaoAplicar
   Kind: bkCustom
   Caption: 'Aplicar'
   Height: 50
   Width: 100

10. Insira um segundo componente BitBtn neste formulário e atribua os seguintes valores para as suas principais propriedades:

    Name: BotaoFechar
    Kind: bkClose
    Caption: 'Fechar'
    Height: 50
    Width: 100

11. Altere a propriedade Name deste formulário para FormCS.

Seu formulário deverá ficar com o aspecto mostrado na figura a seguir.

Figura 28.2: Aspecto inicial do Formulário.

É importante destacar que este formulário será o mesmo em todos os exemplos apresentados neste capítulo. A diferença estará presente nos componentes de acesso a dados, que irão variar de um mecanismo para outro.

12. Insira um componente Database neste formulário e atribua os seguintes valores para as suas principais propriedades:

    AliasName: AXCEL
    Databasename: TESTE
    Connected: True
    Name: DatabaseCS

13. Insira um componente Table neste formulário e atribua os seguintes valores para as suas principais propriedades:

    Databasename: TESTE
    TableName: EMPREGADOS
    Active: True
    Name: TableCS

14. Defina como TableCS o valor da propriedade Dataset do componente DatasorceCS.

15. Insira um componente StoredProc neste formulário e atribua os seguintes valores para as suas principais propriedades:

    Databasename: TESTE
    StoredProcName: AUMENTO
    Name: StoredProcCS

Se você definir como True a propriedade Active deste componente, será exibida uma mensagem de erro, pois não se trata de uma Stored Procedure relacionada a uma consulta.

16. Selecione a propriedade Params do componente StoredProcCS para exibir a caixa de diálogo para definição de parâmetros mostrada na figura a seguir. Nessa caixa de diálogo selecione o parâmetro FATOR (o único disponível) e, no Object Inspector, atribua os seguintes valores às suas principais propriedades: DataType: Float e ParamType: ptInput.

Figura 28.3: Alterando a propriedade Params do componente StoredProcCS.

17. Defina da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoAplicar:

```
procedure TFormCS.BotaoAplicarClick(Sender: TObject);
var
   valor : double;
begin
     try
       valor := StrToFloat(EditValor.Text);
       if valor > 1.0
       then
           begin
               StoredProcCS.ParamByName('Fator').AsString := EditValor.Text;
               StoredProcCS.Prepare;
               StoredProcCS.ExecProc;
               TableCS.Refresh;
           end
       else
           ShowMessage('Valor digitado deve ser maior do que 1.0');
       except
             ShowMessage('Valor Digitado Incompatível');
       end;
end;
```

18. Defina da seguinte maneira o procedimento associado ao evento OnCreate do formulário:

```
procedure TFormCS.FormCreate(Sender: TObject);
begin
  DecimalSeparator := '.';
  DatabaseCS.StartTransaction;
end;
```

19. Defina da seguinte maneira o procedimento associado ao evento OnClose do formulário

```
procedure TFormCS.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  DatabaseCS.Commit;
end;
```

Acessando Banco de Dados do Interbase em uma Aplicação Delphi via Interbase Express

O mecanismo de acesso Interbase Express, cujos componentes estão disponíveis na página Interbase da paleta de componentes, possui um componente chamado IBDatabase que exerce um papel semelhante ao que o componente Database realiza em aplicações que acessam o Interbase via BDE.

A principal diferença é que, no caso do Interbase Express, os componentes IBTable e IBQuery não podem prescindir do componente IBDatabase, ao qual farão referência através da sua propriedade Database.

Para executar uma stored procedure a partir de uma aplicação desenvolvida em Delphi através do mecanismo de acesso Interbase Express, você deve utilizar o componente IBStoredProc, existente na página Interbase da paleta de componentes.

Apresentam-se a seguir as principais propriedades, métodos e eventos desse componente, derivado por herança da classe TIBCustomDataset.

Principais Propriedades do Componente IBStoredProc

Apresentam-se a seguir as principais propriedades desse componente, além daquelas herdadas das suas classes ancestrais.

Database

Essa propriedade, herdada da classe ancestral TIBCustomDataset, é definida como um objeto da classe TIBDatabase, e permite o acesso às propriedades e métodos do componente que representa o banco de dados ao qual se está conectado.

ParamCount

Essa propriedade é definida como uma variável do tipo Word, e define o número de parâmetros da stored procedure que será executada pelo componente, parâmetros estes que podem ser visualizados através da propriedade Params do componente.

Params

Essa propriedade é definida como uma array de objetos da classe TParams, que representam individualmente os parâmetros definidos para a stored procedure. Para acessar a caixa de diálogo na qual são definidos esses parâmetros, basta selecionar, com o botão esquerdo do mouse, as reticências exibidas à direita do nome da propriedade no Object Inspector.

Prepared

Essa propriedade é definida como uma variável booleana, e define se a stored procedure foi preparada para ser executada, de modo a melhorar a sua performance.

A preparação de uma stored procedure pode ser feita atribuindo-se o valor True a essa propriedade, ou mediante uma chamada ao seu método Prepare.

StoredProcName

Essa propriedade é definida como uma variável do tipo string, e define o nome da stored procedure a ser executada.

As stored procedures disponíveis para o banco de dados podem ser selecionadas na lista drop-down exibida à direita do nome da propriedade.

Transaction

Essa propriedade é definida como um objeto da classe TIBTransaction, e define o componente IBTransaction ao qual está vinculado.

Principais Métodos do Componente IBStoredProc

Apresentam-se a seguir os principais métodos desse componente, além daquelas herdadas das suas classes ancestrais.

CopyParams

Declaração

```
procedure CopyParams(Value: TParams);
```

Esse método copia a lista de parâmetros do componente para outra lista de parâmetros (que pode ser a propriedade Params de outro componente StoredProc).

ExecProc

Declaração

```
procedure ExecProc;
```

Esse procedimento executa no servidor a procedure associada ao componente, e cujo nome é definido na sua propriedade StoredProcName.

ParamByName

Declaração

```
function ParamByName(const Value: string): TParam;
```

Esse método permite o acesso individual a parâmetros definidos em uma stored procedure, sendo o nome de um parâmetro passado na forma de uma string na chamada ao procedimento.

Prepare

Declaração

```
procedure Prepare;
```

Esse método prepara a stored procedure a ser executada, atribuindo o valor True à sua propriedade Prepared.

UnPrepare

Declaração

```
procedure UnPrepare;
```

Esse método cancela a preparação da stored procedure a ser executada, atribuindo o valor False à sua propriedade Prepared.

Principais Eventos do Componente IBStoredProc

Apresentam-se a seguir os principais eventos deste componente, além daqueles herdados das suas classes ancestrais.

OnUpdateError

Esse evento ocorre quando um erro é gerado ao se tentar atualizar alterações pendentes no banco de dados.

OnUpdateRecord

Esse evento ocorre ao se atualizar alterações pendentes no registro corrente.

Exemplo de Aplicação

Para recriar a aplicação anterior usando os componentes do Interbase Express (situados na página Interbase da paleta de componentes) você deve executar os seguintes procedimentos:

1. Remova os componentes Database, Storedproc e Table do formulário.

2. Inclua um componente IBDatabase no formulário e usando a sua propriedade DatabaseName, localize o arquivo Exemplo.gdb (repare que neste caso não se utiliza quaisquer alias).

3. Atribua o valor True a sua propriedade Connected. Será exibida a caixa de diálogo Database Login. Preencha corretamente os valores dos campos UserName e Password e o botão Ok, para fechar esta caixa de diálogo e realizar a conexão. Redefina a propriedade Name deste componente para IBDatabaseCS.

4. Coloque um componente IBTransaction no formulário e defina o valor da sua propriedade DefaultDatabase como sendo o nome do componente IBDatabase recém-incluído no formulário (IBDatabaseCS).

5. Defina o valor da propriedade Active deste componente IBTransaction como True. Redefina sua propriedade name como IBTransactionCS.

6. Coloque um componente IBTable no formulário e defina o valor da sua propriedade Database como sendo o nome do componente IBDatabase recém-incluído no formulário (IBDatabaseCS). Redefina sua propriedade name como IBTableCS.

7. Defina o valor da sua propriedade Transaction como sendo o nome do componente IBTransaction inserido no formulário (IBTransactionCS).

8. Defina o valor da sua propriedade TableName como sendo Empregados.

9. Defina o valor da sua propriedade Active como sendo igual a True.

10. Defina o valor da propriedade Dataset do componente DataSource como sendo o nome do componente Table inserido no formulário (IBTableCS).

Repare que, no caso do Interbase Express, os componentes IBTable e IBQuery devem se conectar obrigatoriamente a dois componentes: Um componente IBDatabase e um componente IBTransaction.

O componente IBTransaction permite um controle discreto sobre diferentes transações aplicadas ao banco de dados. Sua propriedade Default Action define como será processada a transação representada pelo componente, podendo assumir um dos seguintes valores:

taRollback: Efetua um RollBack sobre a transação.

taCommit: Efetua um Commit sobre a transação.

taRollbackRetaining: Cancela a transação mas mantém localmente as alterações.

taCommitRetaining: Efetua um Commit sobre a transação mas mantém localmente as alterações.

Repare que, no caso do Interbase Express, os componentes IBTransaction (E não o componente IBDatabase, como poderíamos pensar inicialmente) são responsáveis pelo controle das transações. A única função do componente IBDatabase é realizar a conexão – o controle de transações fica por conta de componentes intransaction. Uma visão mais detalhada dos componentes será apresentada no final do livro – na seção Referências.

11. Redefina como IBTableCS o valor da propriedade Dataset do componente DatasorceCS.

12. Insira um componente IBStoredProc neste formulário e atribua os seguintes valores para as suas principais propriedades:

    Database: IBDatabaseCS
    StoredProcName: AUMENTO
    Name: IBStoredProcCS
    Transaction: IBTransactionCS

13. Selecione a propriedade Params do componente IBStoredProcCS para exibir a caixa de diálogo para definição de parâmetros. Nessa caixa de diálogo selecione o parâmetro FATOR (o único disponível) e, no Object Inspector, atribua os seguintes valores às suas principais propriedades: DataType: Float e ParamType: ptInput.

14. Defina da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoAplicar:

```
procedure TFormCS.BotaoAplicarClick(Sender: TObject);
var
   valor : double;
begin
     try
       valor := StrToFloat(EditValor.Text);
       if valor > 1.0
       then
           begin
               IBStoredProcCS.ParamByName('Fator').AsString := EditValor.Text;
               IBStoredProcCS.Prepare;
               IBStoredProcCS.ExecProc;
               IBTableCS.Refresh;
           end
       else
           ShowMessage('Valor digitado deve ser maior do que 1.0');
       except
             ShowMessage('Valor Digitado Incompatível');
       end;
end;
```

15. Defina da seguinte maneira o procedimento associado ao evento OnCreate do formulário:

```
procedure TFormCS.FormCreate(Sender: TObject);
begin
  DecimalSeparator := '.';
  IBTransactionCS.StartTransaction;
end;
```

16. Defina da seguinte maneira o procedimento associado ao evento OnClose do formulário:

```
procedure TFormCS.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  IBTransactionCS.Commit;
end;
```

Acessando Banco de Dados do Interbase em uma Aplicação Delphi via DBExpress

O mecanismo de acesso DBExpress, cujos componentes estão disponíveis na página DBExpress da paleta de componentes, possui um componente chamado SQLConnection, já visto ao longo do livro, que exerce um papel semelhante ao que o componente Database realiza em aplicações que acessam o Interbase via BDE.

No caso do DBExpress, os componentes SQLTable, SQLQuery e SimpleDataset não podem prescindir do componente SQLConnection, ao qual farão referência através da sua propriedade Connection.

Para executar uma stored procedure a partir de uma aplicação desenvolvida em Delphi através do mecanismo de acesso DBExpress, você deve utilizar o componente SQLStoredProc, existente na página dbExpress da paleta de componentes.

Apresentam-se a seguir as principais propriedades, métodos e eventos desse componente, derivado por herança da classe TCustomSQLDataset.

Principais Propriedades do Componente SQLStoredProc

Apresentam-se a seguir as principais propriedades desse componente, além daquelas herdadas das suas classes ancestrais.

Params

Essa propriedade é definida como uma array de objetos da classe TParams, que representam individualmente os parâmetros definidos para a stored procedure. Para acessar a caixa de diálogo na qual são definidos esses parâmetros, basta selecionar, com o botão esquerdo do mouse, as reticências exibidas à direita do nome da propriedade no Object Inspector.

Prepared

Essa propriedade é definida como uma variável booleana, e define se a stored procedure foi preparada para ser executada, de modo a melhorar a sua performance.

A preparação de uma stored procedure pode ser feita atribuindo-se o valor True a essa propriedade, ou mediante uma chamada ao seu método Prepare.

SQLConnection

Essa propriedade, herdada da classe ancestral TCustomSQLDataset, é definida como um objeto da classe TSQLConnection, e permite o acesso às propriedades e métodos do componente que representa o banco de dados ao qual se está conectado.

StoredProcName

Essa propriedade é definida como uma variável do tipo string, e define o nome da stored procedure a ser executada.

As stored procedures disponíveis para o banco de dados podem ser selecionadas na lista drop-down exibida à direita do nome da propriedade.

Principais Métodos do Componente SQLStoredProc

Apresentam-se a seguir os principais métodos desse componente, além daqueles herdados das suas classes ancestrais.

ExecProc

Declaração

```
procedure ExecProc;
```

Esse procedimento executa no servidor a procedure associada ao componente, e cujo nome é definido na sua propriedade StoredProcName.

Principais Eventos do Componente SQLStoredProc

Este componente não implementa nenhum evento adicional, além daqueles herdados das suas classes ancestrais.

Exemplo de Aplicação

Para recriar a aplicação anterior usando os componentes do DBExpress (situados na página dBExpress da paleta de componentes) você deve executar os seguintes procedimentos:

1. Remova os componentes IBDatabase, IBTransaction, IBStoredproc e IBTable do formulário.

2. Inclua um componente SQLConnection no formulário, crie uma nova conexão e configure suas propriedades para acessar o arquivo Exemplo.gdb (repare que neste caso também não se utiliza qualquer alias).

3. Atribua o valor True a sua propriedade Connected. Será exibida a caixa de diálogo Database Login. Preencha corretamente os valores dos campos UserName e Password e o botão Ok, para

fechar esta caixa de diálogo e realizar a conexão. Redefina a propriedade Name deste componente para SQLConnectionCS.

4. Coloque um componente SimpleDataset no formulário e defina o valor da sua propriedade Connection como sendo o nome do componente SQL Connection recém-incluído no formulário (SQLConnectionCS). Redefina sua propriedade name como SimpleDatasetCS.

5. Altere o valor da subpropriedade CommandType da propriedade Dataset deste componente para ctTable.

6. Altere o valor da subpropriedade CommandText da propriedade Dataset deste componente para EMPREGADOS.

7. Defina o valor da sua propriedade Active como sendo igual a True.

8. Defina o valor da propriedade Dataset do componente DataSource como sendo o nome do componente SimpleDataset inserido no formulário (SimpleDataset CS).

Repare que, no caso do Interbase Express, os componentes IBTable e IBQuery devem se conectar obrigatoriamente a dois componentes: Um componente IBDatabase e um componente IBTransaction.

O componente IBTransaction permite um controle discreto sobre diferentes transações aplicadas ao banco de dados. Sua propriedade Default Action define como será processada a transação representada pelo componente, podendo assumir um dos seguintes valores:

taRollback: Efetua um RollBack sobre a transação.

taCommit: Efetua um Commit sobre a transação.

taRollbackRetaining: Cancela a transação mas mantém localmente as alterações.

taCommitRetaining: Efetua um Commit sobre a transação mas mantém localmente as alterações.

Repare que, no caso do Interbase Express, os componentes IBTransaction (E não o componente IBDatabase, como poderíamos pensar inicialmente) são responsáveis pelo controle das transações. A única função do componente IBDatabase é realizar a conexão – o controle de transações fica por conta de componentes intransaction. Uma visão mais detalhada dos componentes será apresentada no final do livro – na seção Referências.

9. Redefina como SimpleDataset CS o valor da propriedade Dataset do componente DatasorceCS.

10. Insira um componente SQLStoredProc neste formulário e atribua os seguintes valores para as suas principais propriedades:

    SQLConnection: SQLConnection CS
    StoredProcName: AUMENTO
    Name: SQLStoredProcCS

11. Selecione a propriedade Params do componente IBStoredProcCS para exibir a caixa de diálogo para definição de parâmetros. Nessa caixa de diálogo selecione o parâmetro FATOR (o único disponível) e, no Object Inspector, atribua os seguintes valores às suas principais propriedades: DataType: Float e ParamType: ptInput.

12. Declare, na seção var da unit, uma variável chamada TD do tipo TTransactionDesc, como a seguir:

```
var
  FormCS: TFormCS;
  TD : TTransactionDesc;
```

13. Defina da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoAplicar:

```
procedure TFormCS.BotaoAplicarClick(Sender: TObject);
var
   valor : double;
begin
     try
       valor := StrToFloat(EditValor.Text);
       if valor > 1.0
       then
           begin
               SQLStoredProcCS.ParamByName('Fator').AsString := EditValor.Text;
               SQLStoredProcCS.ExecProc;
               SimpleDatasetCS.Refresh;
           end
       else
           ShowMessage('Valor digitado deve ser maior do que 1.0');
       except
             ShowMessage('Valor Digitado Incompatível');
       end;
end;
```

14. Defina da seguinte maneira o procedimento associado ao evento OnCreate do formulário:

```
procedure TFormCS.FormCreate(Sender: TObject);
begin
  DecimalSeparator := '.';
  TD.TransactionID := 1;
  TD.IsolationLevel := xilREADCOMMITTED;
  SQLConnectionCS.StartTransaction(TD);
end;
```

15. Defina da seguinte maneira o procedimento associado ao evento OnClose do formulário

```
procedure TFormCS.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  SQLConnectionCS.Commit(TD);
end;
```

Repare que, no caso do mecanismo dbExpress, deve-se passar como parâmetro nas chamadas aos métodos StartTransaction e Commit uma variável do tipo composto TTransactionDesc, que identifica uma transação (e desempenha um papel semelhante ao do IBTransaction no Interbase Express).

Capítulo 29

Programação Gráfica

Uma das principais diferenças entre o sistema operacional Windows e o antigo DOS é que, enquanto este é baseado em caracteres, aquele constitui uma interface gráfica em que até mesmo os caracteres são desenhados. O Windows trata a tela (e suas regiões) e áreas de impressão como uma superfície de desenho, também conhecida como "Canvas".

As funções relacionadas às atividades de desenho do Windows estão reunidas em um grupo ou biblioteca de funções denominado GDI – Graphical Device Interface. A utilização dessas funções normalmente requer um número que identifica o objeto sobre o qual a função vai atuar, e esse número costuma ser denominado "handle" do objeto.

A fim de facilitar o acesso a estas funcionalidades, o Delphi definiu uma nova classe denominada TCanvas, que incorporou entre seus métodos as principais funções da GDI. Ao definir objetos da classe TCanvas como propriedades ou campos de outras classes e componentes, o Delphi eliminou a necessidade de utilização de "handles" e chamada das funções da GDI – estes serão necessários apenas quando forem utilizadas funções da GDI existentes na API do Windows, mas que não foram incorporadas como métodos da classe TCanvas. Nesse caso, no entanto, a utilização dessas funções da GDI ainda será bastante simplificada, pois o Delphi definiu para esses componentes uma propriedade chamada handle, que retorna o código que identifica o objeto.

KNOW-HOW em: Definição de Desenhos em Run-Time

PRÉ-REQUISITOS

- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com Delphi.
- Conhecimento das principais propriedades e métodos das classes TCanvas, TBrush e TPen.

METODOLOGIA

- Apresentação do problema: Utilização das principais propriedades e métodos das classes TCanvas, TBrush e TPen na definição de desenhos em run-time para aplicações desenvolvidas em Delphi.

TÉCNICA

- Apresentação dos procedimentos necessários à definição de desenhos em run-time para aplicações desenvolvidas em Delphi.

A Classe TCanvas

Para o Windows, qualquer área a ser manipulada na tela (ou na impressora) é tratada como uma superfície de desenho.

Para simplificar a vida do programador, o Delphi criou uma nova classe, denominada TCanvas, que representa uma superfície de desenho retangular sobre a qual poderão ser feitos desenhos, textos, etc.

Em um canvas existe um sistema de coordenadas cartesiano em que a origem está situada no canto superior esquerdo, as abscissas crescem da esquerda para a direita e as ordenadas de cima para baixo (ao contrário do sistema cartesiano convencional). Conforme será visto posteriormente, existem funções da GDI do Windows que permitem que se altere a orientação destes eixos do canvas, bem como o posicionamento da origem desse sistema.

Guarde bem esse conceito: um canvas é um objeto da classe TCanvas que define uma área de desenho da tela, e que tem, entre seus métodos, alternativas mais práticas às principais funções da GDI do Windows.

O desenho de linhas em um canvas é feito usando-se uma caneta imaginária, que é na realidade um objeto da classe TPen (outra classe definida pelo Delphi), e o preenchimento de superfícies é feito usando-se um pincel imaginário, que é na realidade um objeto da classe TBrush. Dentre as principais propriedades de um objeto da classe TPen – a caneta imaginária – destacam-se aquelas que definem a sua espessura, a sua cor, modo e estilo de desenho. Para um objeto da classe TBrush, suas principais propriedades são aquelas que definem a sua cor e estilo.

As principais propriedades e métodos dessas classes foram apresentados anteriormente e não serão redefinidos neste capítulo, no qual trataremos apenas da sua utilização em algumas situações particulares.

O Componente Shape

O componente Shape, situado na página Additional da paleta de componentes, é utilizado para desenhar formas geométricas na tela e, diferentemente de outros componentes, permite que suas propriedades Brush e Pen sejam definidas durante o projeto da aplicação alterando os seus valores diretamente no Object Inspector.

Existem componentes, no entanto, que não permitem que se acessem suas propriedades Brush e Pen na fase de projeto, mas apenas durante a execução do aplicativo (ou seja, via código).

Esses componentes, no entanto, têm um evento chamado OnPaint, para o qual podemos definir um procedimento associado, e nesse procedimento definir o código referente a qualquer desenho a ser feito no seu canvas.

Para que um desenho seja permanentemente visualizado em um formulário ou componente, o código que o define deve ser digitado no procedimento associado ao evento OnPaint do formulário ou componente. Se esse código não for incluído no evento OnPaint e a região do formulário ou componente que exibe o desenho for sobreposta por outra janela, este não será restaurado na tela quando a região correspondente se tornar visível novamente.

Nos próximos tópicos, serão apresentados os procedimentos necessários à exibição de desenhos em formulários.

Desenhando em um Formulário

Todo formulário tem uma propriedade chamada canvas que é, como já foi dito anteriormente, um objeto da classe TCanvas. Dessa maneira, se quisermos desenhar sobre a superfície de um formulário, devemos fazê-lo utilizando a sua propriedade canvas.

Apresentamos a seguir um aplicativo bastante simples, que permite que o usuário defina os pontos que delimitam um retângulo ou elipse, bem como as suas dimensões.

Para criar esse aplicativo, você deve executar os seguintes procedimentos:

1. Selecionar o item New Application do menu File, para iniciar um novo projeto de aplicação.
2. Atribua os valores a seguir para as principais propriedades do Formulário Principal da Aplicação:

   Name: FormDesenho
   Width: 870
   Height: 640
   Caption: Exemplo de Utilização do Canvas de um Formulário
3. Inclua um componente panel (página Standard) no formulário e atribua os seguintes valores às suas principais propriedades:

   Name: Panel1
   Height: 150
   Align: alTop
   Caption:
4. Inclua um componente GroupBox (página Standard) no panel e atribua os seguintes valores para as suas principais propriedades:

   Name: GroupBox1
   Left: 8
   Top: 8
   Width: 260
   Height: 75
   Caption: Coordenadas iniciais
5. Inclua dois Labels no componente GroupBox1 e atribua os seguintes valores para as suas principais propriedades:

   Name: Label1
   Left: 17
   Top: 22
   Height: 13
   AutoSize: True
   Caption: X:

   Name: Label2
   Left: 17
   Top: 50
   Height: 13
   AutoSize: True
   Caption: Y:
6. Inclua dois componentes Edit no componente GroupBox1 e atribua os seguintes valores para as suas principais propriedades:

   Name: EditX
   Left: 41

Top: 18
Width: 40
Height: 21
Text: 10

Name: EditY
Left: 41
Top: 42
Width: 40
Height: 21
Text: 10

7. Inclua um segundo componente GroupBox no panel e atribua os seguintes valores para as suas principais propriedades:

   Name: GroupBox2
   Left: 144
   Top: 8
   Width: 129
   Height: 73
   Caption: Dimensões Iniciais

8. Inclua dois Labels no componente GroupBox2 e atribua os seguintes valores para as suas principais propriedades:

   Name: Label3
   Left: 13
   Top: 22
   Height: 13
   AutoSize: True
   Caption: Largura:

   Name: Label4
   Left: 13
   Top: 48
   Height: 13
   AutoSize: True
   Caption: Altura:

9. Inclua dois componentes Edit no componente GroupBox2 e atribua os seguintes valores para as suas principais propriedades:

   Name: EditW
   Left: 62
   Top: 18
   Width: 40
   Height: 21
   Text: 100

Name: EditH
Left: 62
Top: 44
Width: 40
Height: 21
Text: 100

10. Inclua um componente RadioGroup (página Standard) no panel e atribua os seguintes valores para as suas principais propriedades:

    Name: RadioGroupForma
    Left: 280
    Top: 8
    Width: 129
    Height: 73
    Caption: Forma
    ItemIndex: 0

11. Defina a propriedade Items do componente RadioGroupForma usando a caixa de diálogo String List Editor, como indicado na figura a seguir.

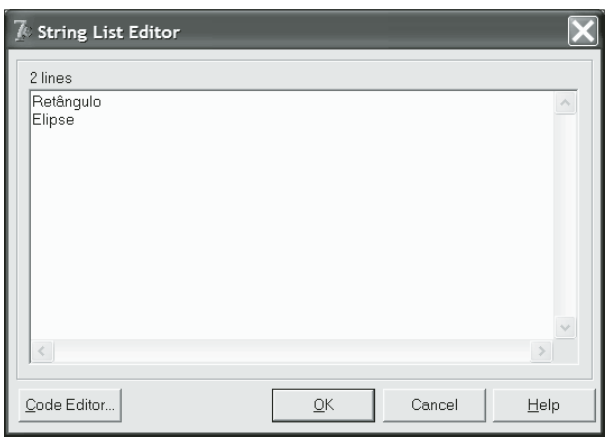


Figura 29.1: Definindo a propriedade Items para o RadioGroup.

12. Inclua um componente BitBtn (página Additional) no panel e atribua os seguintes valores para as suas principais propriedades:

    Name: BotaoDesenho
    Left: 424
    Top: 20
    Width: 100
    Height: 50
    Caption: Desenha
    Kind: bkOK

13. Inclua mais dois Labels no panel e atribua os seguintes valores para as suas principais propriedades:

    Name: Label5
    Left: 16
    Top: 88
    Width: 120
    Height: 13
    Caption: Estilo de Preenchimento:

    Name: Label6
    Left: 184
    Top: 88
    Width: 22
    Height: 13
    Caption: Cor:

14. Inclua um componente ComboBox no panel e atribua os seguintes valores para as suas principais propriedades:

    Name: ComboBoxEstilo
    Left: 16
    Top: 112
    Width: 145
    Height: 21
    Style: csDropDownList
    ItemHeight: 13

15. Defina a propriedade Items do componente ComboBoxEstilo usando a caixa de diálogo String List Editor, como indicado na figura a seguir.

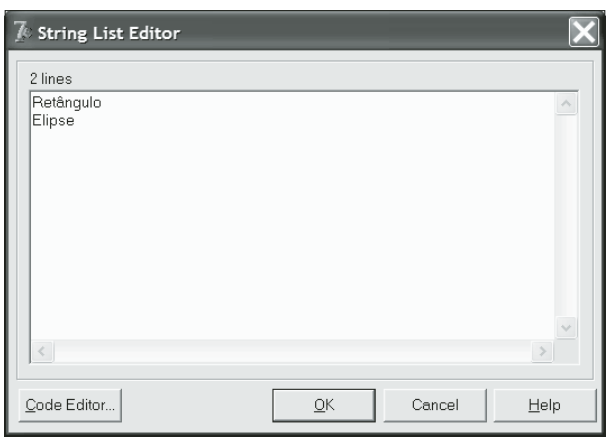


Figura 29.2: Definindo a propriedade Items para o ComboBox.

16. Inclua um componente Shape (página Additional) no panel e atribua os seguintes valores para as suas principais propriedades:

    Name: ShapeCor
    Left: 184
    Top: 104
    Width: 41
    Height: 33
    Brush.Color: clWhite
    Brush.Style: bsSolid
    Shape: stRectangle

17. Inclua um componente ColorDialog (página Dialogs) no panel e mantenha os valores default para as suas propriedades.

A figura a seguir apresenta o aspecto final do formulário.

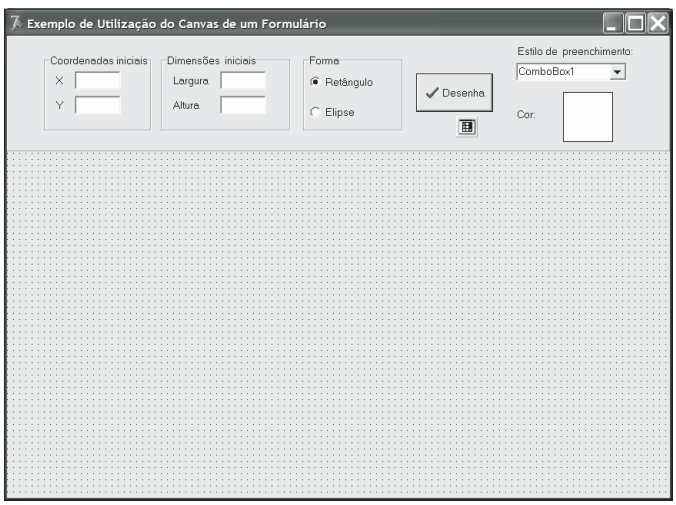


Figura 29.3: Aspecto do formulário.

18. Defina as seguintes variáveis na seção var do arquivo de código associado ao formulário:

```
x, y, h, w : integer;
```

19. Defina da seguinte maneira o procedimento associado ao evento OnClick do componente BotaoDesenha:

```
procedure TFormDesenho.BotaoDesenhoClick(Sender: TObject);
begin
     FormDesenho.Canvas.Brush.Color := ShapeCor.Brush.Color;
     FormDesenho.Canvas.Brush.Style := TBrushStyle(ComboBoxEstilo.ItemIndex);
     x := StrToInt(EditX.Text);
     y := StrToInt(Edity.Text)+149;
     h := StrToInt(Edith.Text);
     w := StrToInt(Editw.Text);
     if (x<0)or(x+w>FormDesenho.Width)or(Y<0)or(Y+h>FormDesenho.Height)
```

```
        then
            ShowMessage('Parametros Incorretos')
        else
            FormDesenho.Repaint;
end;
```

Este código realiza as seguintes tarefas:

- Define como valor da propriedade Color da subpropriedade Brush do canvas do formulário o valor definido na subpropriedade Color da propriedade Brush do componente ShapeCor, o que é feito na seguinte linha de código:

```
FormDesenho.Canvas.Brush.Color := ShapeCor.Brush.Color;
```

- Define como valor da propriedade Style da subpropriedade Brush do canvas do formulário o valor selecionado no componente ComboBoxEstilo.

```
FormDesenho.Canvas.Brush.Style := TBrushStyle(ComboBoxEstilo.ItemIndex);
```

- Atribui às variáveis X, Y, H e W os valores definidos nas caixas de texto correspondentes, como indicado no trecho de código a seguir:

```
x := StrToInt(EditX.Text);
y := StrToInt(Edity.Text)+149;
h := StrToInt(Edith.Text);
w := StrToInt(Editw.Text);
```

Repare que ao valor de y foi acrescido o valor da altura do panel, para que a figura não seja oculta pelo mesmo.

- Verifica se os valores definidos estão dentro de uma faixa que garanta a visualização da figura, o que é feito no seguinte trecho de código:

```
if (x<0)or(x+w>FormDesenho.Width)or(Y<0)or(Y+h>FormDesenho.Height)
   then
       ShowMessage('Parametros Incorretos')
   else
       FormDesenho.Repaint;
```

Caso os valores estejam dentro da faixa permitida, o método Repaint do formulário faz com que o seu procedimento associado ao evento OnPaint seja executado.

20. Compartilhe esse procedimento com os seguintes eventos:

    Evento OnShow do Formulário.
    Evento OnChange do Componente EditX.
    Evento OnChange do Componente EditY.
    Evento OnChange do Componente EditH.
    Evento OnChange do Componente EditW.
    Evento OnClick do Componente RadioGroupForma.
    Evento OnChange do Componente ComboBoxEstilo.

Isso garante que o efeito de qualquer alteração feita nos valores que definem a figura seja imediatamente visualizado. Garante também que a figura default seja visualizada quando o formulário for exibido pela primeira vez.

21. Defina da seguinte maneira o procedimento associado ao evento OnMouseDown do componente ShapeCor:

```
procedure TFormDesenho.ShapeCorMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     if ColorDialog1.Execute then
     begin
        ShapeCor.Brush.Color := ColorDialog1.Color;
        BotaoDesenhoClick(Self);
     end;
end;
```

Esse procedimento exibe uma caixa de diálogo de seleção de cores para que o usuário escolha uma nova cor para a figura.

Caso o método Execute do componente OpenDialog1 retorne o valor True, o procedimento associado ao evento OnClick do componente BotaoDesenho é executado.

22. Defina da seguinte maneira o procedimento associado ao evento OnPaint do formulário:

```
procedure TFormDesenho.FormPaint(Sender: TObject);
begin
    if RadioGroupForma.ItemIndex = 0
    then
          FormDesenho.Canvas.Rectangle(x,y,x+w,y+h)
    else
          FormDesenho.Canvas.Ellipse(x,y,x+w,y+h);
end;
```

Esse procedimento verifica o tipo de figura selecionada pelo usuário (analisando o valor da propriedade ItemIndex do componente RadioGroupForma) e executa o método de desenho correspondente.

23. Defina da seguinte maneira o procedimento associado ao evento OnCreate do formulário:

```
procedure TFormDesenho.FormCreate(Sender: TObject);
begin
     ComboBoxEstilo.ItemIndex := 0;
end;
```

24. Salve a unidade de código associada a esse formulário com o nome UnitDesenha.

Apresentamos a seguir a unidade de código associada ao formulário:

```
unit UnitDesenha;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ExtCtrls;

type
  TFormDesenho = class(TForm)
    Panel1: TPanel;
    GroupBox1: TGroupBox;
    Label1: TLabel;
```

```
    EditX: TEdit;
    Label2: TLabel;
    EditY: TEdit;
    GroupBox2: TGroupBox;
    Label3: TLabel;
    EditH: TEdit;
    Label4: TLabel;
    EditW: TEdit;
    RadioGroupForma: TRadioGroup;
    BotaoDesenho: TBitBtn;
    Label5: TLabel;
    ComboBoxEstilo: TComboBox;
    Label6: TLabel;
    ShapeCor: TShape;
    ColorDialog1: TColorDialog;
    procedure BotaoDesenhoClick(Sender: TObject);
    procedure ShapeCorMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormPaint(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormDesenho: TFormDesenho;
  x, y, h, w : integer;

implementation

{$R *.DFM}

procedure TFormDesenho.BotaoDesenhoClick(Sender: TObject);
begin
     FormDesenho.Canvas.Brush.Color := ShapeCor.Brush.Color;
     FormDesenho.Canvas.Brush.Style := TBrushStyle(ComboBoxEstilo.ItemIndex);
     x := StrToInt(EditX.Text);
     y := StrToInt(Edity.Text)+149;
     h := StrToInt(Edith.Text);
     w := StrToInt(Editw.Text);
     if (x<0)or(x+w>FormDesenho.Width)or(Y<0)or(Y+h>FormDesenho.Height)
     then
         ShowMessage('Parametros Incorretos')
     else
         FormDesenho.Repaint;
end;

procedure TFormDesenho.ShapeCorMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     if ColorDialog1.Execute then
     begin
        ShapeCor.Brush.Color := ColorDialog1.Color;
        BotaoDesenhoClick(Self);
     end;
end;

procedure TFormDesenho.FormPaint(Sender: TObject);
begin
    if RadioGroupForma.ItemIndex = 0
    then
```

```
            FormDesenho.Canvas.Rectangle(x,y,x+w,y+h)
        else
            FormDesenho.Canvas.Ellipse(x,y,x+w,y+h);
    end;

    procedure TFormDesenho.FormCreate(Sender: TObject);
    begin
         ComboBoxEstilo.ItemIndex := 0;
    end;

    end.
```

A figura a seguir apresenta o aplicativo sendo executado.

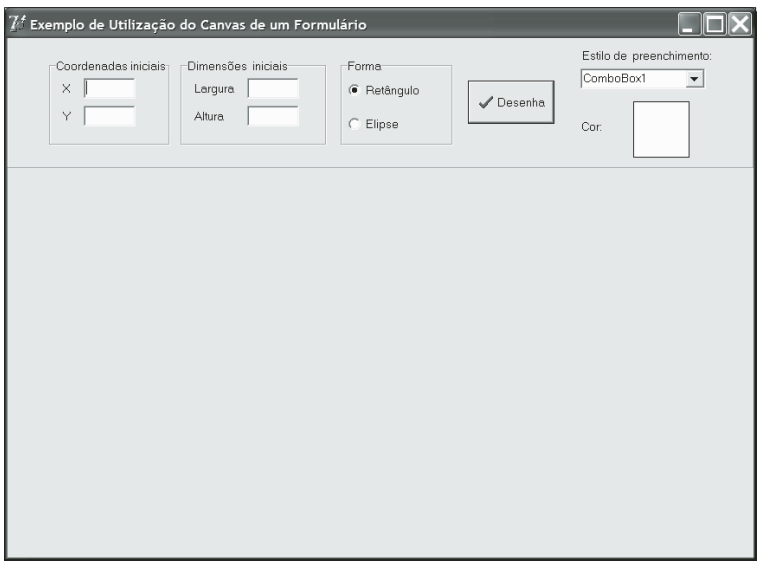


Figura 29.4: Executando o aplicativo.

KNOW-HOW EM: DEFINIÇÃO DE DESENHOS DE FORMA INTERATIVA

PRÉ-REQUISITOS

- Experiência prévia na utilização de componentes no desenvolvimento de aplicações em Delphi.
- Conhecimento das principais propriedades e métodos das classes TCanvas, TBrush e TPen.

METODOLOGIA

- Apresentação do problema: Utilização das principais propriedades e métodos das Classes TCanvas, TBrush e TPen da definição de desenhos em run-time de forma interativa para aplicações desenvolvidas com o Borland Delphi 7.

TÉCNICA

- Apresentação dos procedimentos necessários à definição de desenhos em run-time de forma interativa para aplicações desenvolvidas com o Borland Delphi 7.

Desenhando de Forma Interativa

Neste tópico, mostramos os procedimentos necessários ao desenho interativo em um canvas de um formulário.

Conforme já descrito anteriormente, o desenho de uma linha em um canvas é feito utilizando-se os seus métodos MoveTo (que movimenta a caneta imaginária para a posição desejada) e LineTo (que realmente desenha a linha).

A fim de ilustrar esses procedimentos, vamos criar um pequeno aplicativo que permita o desenho interativo de retas no canvas de um formulário. A lógica desse problema é a seguinte:

- Quando o usuário pressionar o botão esquerdo do mouse, estará indicando que deseja iniciar o desenho de uma reta a partir daquele ponto. Vamos chamar de xbase e ybase as coordenadas desse ponto, que denominaremos ponto-base. O usuário deve manter o botão esquerdo pressionado e movimentar o mouse pela área de desenho. A obtenção dessas coordenadas do ponto-base não é difícil, pois estes valores são passados como parâmetros (denominados X e Y) no procedimento associado ao evento OnMouseDown do formulário.
- Enquanto o mouse estiver se deslocando pela área de desenho, uma linha "elástica" deverá ser desenhada do ponto-base até a posição corrente do mouse. Vamos chamar de xatual e yatual as coordenadas desse ponto. A obtenção das coordenadas do ponto corrente também não é difícil, pois esses valores são passados como parâmetros (denominados X e Y) no procedimento associado ao evento OnMouseMove do formulário.
- Para desenhar a linha, devemos executar o método MoveTo(xbase,ybase), para colocar a caneta na posição do ponto-base, e o Método LineTo(Xatual,Yatual), para desenhar a linha até o ponto corrente.
- Quando o usuário soltar o botão esquerdo do mouse, a linha deverá ser desenhada na forma definitiva. Essa tarefa também não é difícil, pois o procedimento associado ao evento OnMouseUp também passa como parâmetros os valores X e Y das coordenadas atuais do mouse.
- Para evitar que o programa desenhe linhas quando o botão esquerdo não estiver pressionado, devemos criar uma variável booleana que indique se o programa deve ou não executar as funções de desenho.

Para criar essa aplicação, você deve executar os seguintes procedimentos:

1. Selecione o item New Application do menu File para iniciar um novo projeto de aplicação.

2. Atribua os valores a seguir para as principais propriedades do Formulário Principal da Aplicação.

   Name: FormInterativo
   Width: 870
   Height: 640
   Caption: Exemplo de Desenho Interativo no Canvas de um Formulário
   Color: clBlack

3. Inclua um componente Panel no formulário e atribua os seguintes valores às suas principais propriedades:

Name: Panel1
Align: albotton
Height: 40
Caption:

A Figura 29.5 apresenta o aspecto do formulário.

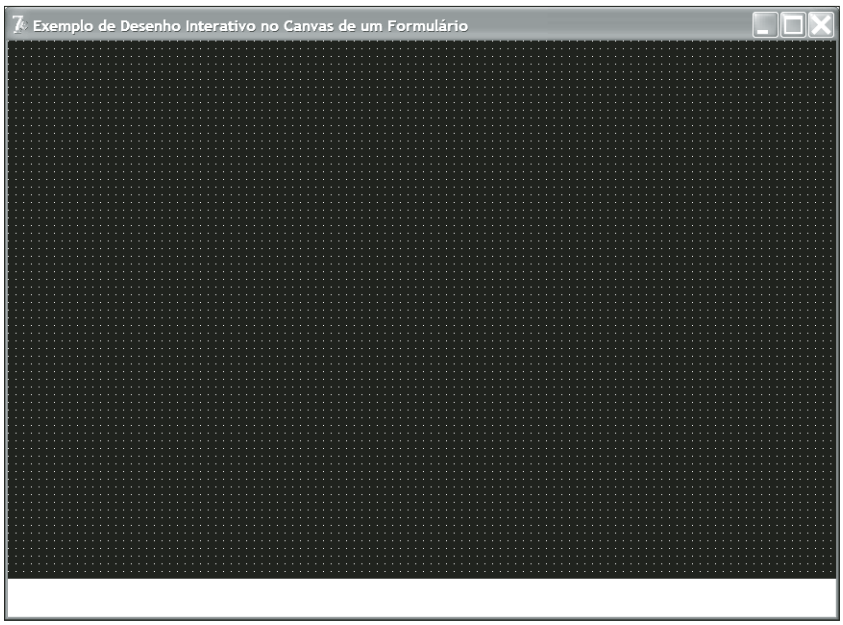


Figura 29.5: Aspecto do formulário.

4. Declare as seguintes variáveis na seção var da unit do formulário:

```
desenha : boolean;
xbase, ybase, xatual, yatual : integer;
```

5. Defina da seguinte maneira o procedimento associado ao evento OnCreate do formulário:

```
procedure TFormInterativo.FormCreate(Sender: TObject);
begin
     desenha := False;
end;
```

Isso garante que a variável desenha seja inicializada com o valor False.

6. Defina da seguinte maneira o procedimento associado ao evento OnMouseDown do formulário:

```
procedure TFormInterativo.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     desenha := True;
     xbase := X;
     ybase := y;
     xatual := X;
     yatual := y;
end;
```

Como pode ser verificado, esse procedimento define os valores das variáveis xbase e ybase. Inicialmente, os mesmos valores são atribuídos às variáveis xatual e yatual.

7. Defina da seguinte maneira o procedimento associado ao evento OnMouseMove do formulário:

```
procedure TFormInterativo.FormMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
Begin
     Panel1.Caption := 'X = '+IntToStr(X)+ ' Y = '+IntToStr(Y);
     If not desenha then exit;
     FormInterativo.Canvas.Pen.Color := clBlack;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
     xatual := x;
     yatual := y;
     FormInterativo.Canvas.Pen.Color := clWhite;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
end;
```

Inicialmente, exibem-se no panel as coordenadas atuais do mouse, o que é feito na seguinte linha de código:

```
Panel1.Caption := 'X = '+IntToStr(X)+ ' Y = '+IntToStr(Y);
```

Em seguida, verifica-se se a variável desenha é igual a True. Se for False, então "not desenha" é True e a procedure não é executada.

Em seguida, atribui-se a cor preta à caneta do canvas para apagar a linha desenhada anteriormente, o que é feito no seguinte trecho de código:

```
FormInterativo.Canvas.Pen.Color := clBlack;
FormInterativo.Canvas.MoveTo(xbase,ybase);
FormInterativo.Canvas.LineTo(xatual,yatual);
```

Por fim, atualizam-se os valores de xatual e yatual, atribui-se a cor branca à caneta do canvas e desenha-se a nova linha, o que é feito no seguinte trecho de código:

```
xatual := x;
yatual := y;
FormInterativo.Canvas.Pen.Color := clWhite;
FormInterativo.Canvas.MoveTo(xbase,ybase);
FormInterativo.Canvas.LineTo(xatual,yatual);
```

8. Defina da seguinte maneira o procedimento associado ao evento OnMouseUp do formulário:

```
procedure TFormInterativo.FormMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     desenha := False;
     FormInterativo.Canvas.Pen.Color := clBlack;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
     xatual := x;
     yatual := y;
     FormInterativo.Canvas.Pen.Color := clWhite;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
end;
```

Esse procedimento atribui o valor False à variável desenha e executa um código de desenho semelhante ao descrito anteriormente.

A figura a seguir apresenta o aplicativo sendo executado.

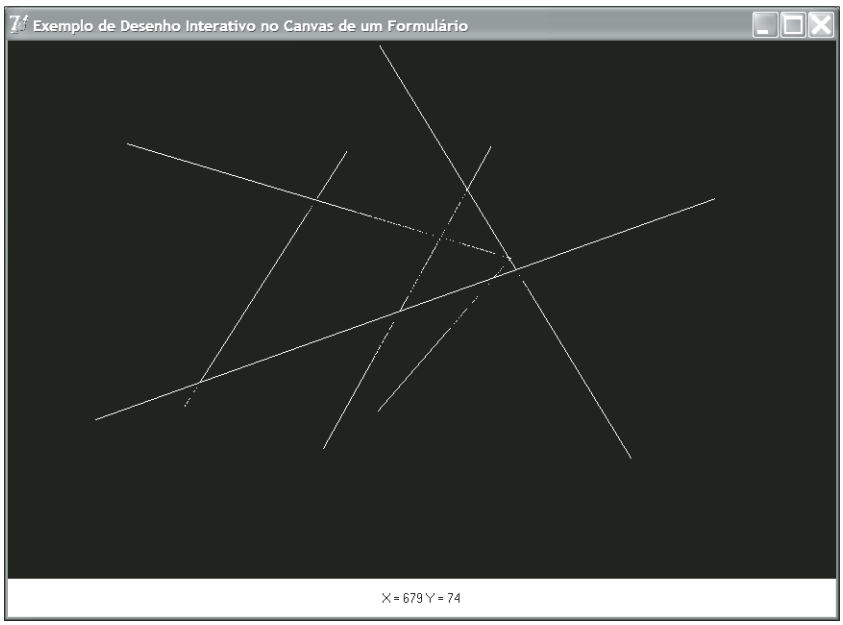


Figura 29.6: Execução do aplicativo.

Apresentamos a seguir a unidade de código associada a esse formulário.

```
unit UnitPaint;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls;

type
  TFormInterativo = class(TForm)
    Panel1: TPanel;
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormInterativo: TFormInterativo;
  desenha : boolean;
```

```
  xbase, ybase, xatual, yatual : integer;

implementation

{$R *.DFM}

procedure TFormInterativo.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
     Panel1.Caption := 'X = '+IntToStr(X)+ ' Y = '+IntToStr(Y);
     If not desenha then exit;
     FormInterativo.Canvas.Pen.Color := clBlack;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
     xatual := x;
     yatual := y;
     FormInterativo.Canvas.Pen.Color := clWhite;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
end;

procedure TFormInterativo.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     desenha := True;
     xbase := X;
     ybase := y;
     xatual := X;
     yatual := y;
end;

procedure TFormInterativo.FormMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     desenha := False;
     FormInterativo.Canvas.Pen.Color := clBlack;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
     xatual := x;
     yatual := y;
     FormInterativo.Canvas.Pen.Color := clWhite;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
end;

procedure TFormInterativo.FormCreate(Sender: TObject);
begin
     desenha := False;
end;
end.
```

Esse aplicativo ainda apresenta, no entanto, alguns problemas:

Se você ocultar parte do formulário, ou minimizá-lo e restaurá-lo em seguida, o desenho será perdido! Isso se deve ao fato de que não foi definido um procedimento associado ao evento OnPaint, e logo nada existe para ser redesenhado, ou que garanta a persistência deste desenho.

Há várias alternativas para resolver este problema, sendo que uma delas consiste em realizar sempre uma cópia do desenho atual em um Bitmap oculto, existente na menória, e copiar a imagem do Bitmap para o formulário quando o evento OnPaint for disparado.

A Classe TBitmap

O Delphi possui a classe TBitmap, que permite que se façam coisas incríveis com Bitmaps. Neste tópico, no entanto, limitaremos nosso estudo a resolver o problema descrito no final do tópico anterior.

A solução consiste em criar um objeto da classe TBitmap com as mesmas dimensões do formulário e, sempre que o desenho do formulário for alterado (nesse caso, quando uma nova reta for incluída), copiar o desenho do formulário para o bitmap. Por outro lado, quando o formulário for redimensionado, ou ocultado e exibido em seguida, devemos copiar o desenho do Bitmap para o formulário.

Assim como um formulário, um objeto da classe TBitmap também tem uma propriedade chamada Canvas (que é, obviamente, um objeto da classe TCanvas).

A Classe TCanvas possui um método chamado CopyRect que permite que se copie para o seu interior uma área retangular de outro canvas. Esse método tem, como parâmetros:

- Uma variável do tipo TRect, representando a área retangular de destino do canvas que chama o método.
- Uma variável do tipo TCanvas, que define o canvas de onde será feita a cópia.
- Uma outra variável do tipo TRect, representando a área retangular de origem do canvas de onde é feita a cópia.

O tipo composto TRect (que não é uma classe, mas um Record) possui os seguintes campos:

- Top: Uma variável inteira, que define a ordenada da extremidade superior esquerda da região retangular.
- Bottom: Uma variável inteira, que define a ordenada da extremidade inferior direita da região retangular.
- Left: Uma variável inteira, que define a abscissa da extremidade superior esquerda da região retangular.
- Right: Uma variável inteira, que define a abscissa da extremidade inferior direita da região retangular.

A região a ser copiada, nesse caso, será definida pela área cliente do formulário, descontada a altura do panel (definida pela sua propriedade Height).

Para resolver este problema, portanto, você deve executar os seguintes procedimentos no aplicativo criado no tópico anterior:

1. Declarar uma variável chamada MeuBitmap, da classe TBitmap, na seção var da unit do formulário, como a seguir:

   ```
   MeuBitmap : TBitmap;
   ```

2. Declarar duas variáveis chamadas AreaDestino e AreaOrigem, do tipo TRect, na seção var da unit do formulário, como a seguir:

   ```
   Areadestino, AreaOrigem : TRect;
   ```

3. Criar a instância da classe TBitmap no procedimento associado ao evento OnCreate do formulário, que será redefinido como mostrado a seguir:

```
procedure TFormInterativo.FormCreate(Sender: TObject);
begin
     desenha := False;
     MeuBitmap := TBitmap.Create;
end;
```

4. Redefinir como a seguir o procedimento associado ao evento OnMouseUp do formulário, incluindo as linhas de código responsáveis pela cópia do desenho definido no canvas do formulário para o canvas do bitmap:

```
procedure TFormInterativo.FormMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     desenha := False;
     FormInterativo.Canvas.Pen.Color := clBlack;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
     xatual := x;
     yatual := y;
     FormInterativo.Canvas.Pen.Color := clWhite;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
     MeuBitmap.Height := FormInterativo.ClientHeight-Panel1.Height;
     MeuBitmap.Width := FormInterativo.ClientWidth;
     Areadestino.Top := 0;
     Areadestino.Left := 0;
     Areadestino.Right := FormInterativo.ClientWidth;
     Areadestino.Bottom := FormInterativo.ClientHeight-Panel1.Height;
     AreaOrigem.Top := 0;
     AreaOrigem.Left := 0;
     AreaOrigem.Right := FormInterativo.ClientWidth;
     AreaOrigem.Bottom := FormInterativo.ClientHeight-Panel1.Height;
 MeuBitmap.Canvas.CopyRect(AreaDestino,FormInterativo.Canvas,AreaOrigem);
end;
```

5. Redefinir como a seguir o procedimento associado ao evento OnPaint do formulário, incluindo as linhas de código responsáveis pela cópia do desenho definido no canvas do bitmap para o canvas do formulário:

```
procedure TFormInterativo.FormPaint(Sender: TObject);
begin
     Areadestino.Top := 0;
     Areadestino.Left := 0;
     Areadestino.Right := FormInterativo.ClientWidth;
     Areadestino.Bottom := FormInterativo.ClientHeight-Panel1.Height;
     AreaOrigem.Top := 0;
     AreaOrigem.Left := 0;
     AreaOrigem.Right := FormInterativo.ClientWidth;
     AreaOrigem.Bottom := FormInterativo.ClientHeight-Panel1.Height;
 FormInterativo.Canvas.CopyRect(AreaDestino,MeuBitmap.Canvas,AreaOrigem);
end;
```

Repare que agora a imagem é transferida no sentido inverso.

6. Compartilhe o procedimento associado ao evento OnPaint do formulário com o seu evento OnResize.

7. Defina da seguinte maneira o procedimento associado ao evento OnDestroy do formulário, para liberar a memória alocada para o objeto MeuBitmap:

```
procedure TFormInterativo.FormDestroy(Sender: TObject);
begin
     MeuBitmap.Free;
end;
```

Apresenta-se a seguir o código desta unit, após terem sido feitas as alterações relacionadas:

```
unit UnitPaint;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls;

type
  TFormInterativo = class(TForm)
    Panel1: TPanel;
    procedure FormCreate(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure FormMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormPaint(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormInterativo: TFormInterativo;
  desenha : boolean;
  xbase, ybase, xatual, yatual : integer;
  MeuBitmap : TBitmap;
  Areadestino, AreaOrigem : TRect;

implementation

{$R *.DFM}

procedure TFormInterativo.FormCreate(Sender: TObject);
begin
     desenha := False;
     MeuBitmap := TBitmap.Create;
end;

procedure TFormInterativo.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     desenha := True;
     xbase := X;
```

```
     ybase := y;
     xatual := X;
     yatual := y;
end;

procedure TFormInterativo.FormMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
     Panel1.Caption := 'X = '+IntToStr(X)+ ' Y = '+IntToStr(Y);
     If not desenha then exit;
     FormInterativo.Canvas.Pen.Color := clBlack;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
     xatual := x;
     yatual := y;
     FormInterativo.Canvas.Pen.Color := clWhite;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
end;

procedure TFormInterativo.FormMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     desenha := False;
     FormInterativo.Canvas.Pen.Color := clBlack;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
     xatual := x;
     yatual := y;
     FormInterativo.Canvas.Pen.Color := clWhite;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
     MeuBitmap.Height := FormInterativo.ClientHeight-Panel1.Height;
     MeuBitmap.Width := FormInterativo.ClientWidth;
     Areadestino.Top := 0;
     Areadestino.Left := 0;
     Areadestino.Right := FormInterativo.ClientWidth;
     Areadestino.Bottom := FormInterativo.ClientHeight-Panel1.Height;
     AreaOrigem.Top := 0;
     AreaOrigem.Left := 0;
     AreaOrigem.Right := FormInterativo.ClientWidth;
     AreaOrigem.Bottom := FormInterativo.ClientHeight-Panel1.Height;
     MeuBitmap.Canvas.CopyRect(AreaDestino,FormInterativo.Canvas,AreaOrigem);
end;

procedure TFormInterativo.FormPaint(Sender: TObject);
begin
     Areadestino.Top := 0;
     Areadestino.Left := 0;
     Areadestino.Right := FormInterativo.ClientWidth;
     Areadestino.Bottom := FormInterativo.ClientHeight-Panel1.Height;
     AreaOrigem.Top := 0;
     AreaOrigem.Left := 0;
     AreaOrigem.Right := FormInterativo.ClientWidth;
     AreaOrigem.Bottom := FormInterativo.ClientHeight-Panel1.Height;
     FormInterativo.Canvas.CopyRect(AreaDestino,MeuBitmap.Canvas,AreaOrigem);
end;

procedure TFormInterativo.FormDestroy(Sender: TObject);
begin
     MeuBitmap.Free;
end;

end.
```

Pronto! Agora seu desenho não será mais perdido quando a janela for minimizada ou sobreposta! Graças ao uso de um objeto da classe TBitmap, garantiu-se a pesistência do desenho.

Modos de Mapeamento

No início deste capítulo, dissemos que as ordenadas crescem de cima para baixo, e você deve ter constatado isso ao analisar os valores de Y quando movimentava o mouse no aplicativo desenvolvido nos tópicos anteriores.

Para alterar as orientações dos eixos, o posicionamento dos eixos e a escala adotada, devem-se utilizar as funções de mapeamento existentes na API do Windows.

As funções da GDI da API do Windows operam com as denominadas coordenadas lógicas, que devem ser convertidas em coordenadas físicas, ou coordenadas da tela.

Embora as coordenadas físicas sejam limitadas pelas dimensões em pixels da área de desenho, nosso problema pode manipular coordenadas em variadas faixas de valores. O importante é que seja possível transformar corretamente as coordenadas lógicas em coordenadas físicas, o que é feito usando as funções de mapeamento de coordenadas da GDI do Windows.

O modo de mapeamento é definido utilizando-se a função SetMapMode da API do Windows. Essa função requer como parâmetros:

- Um handle para o objeto sobre o qual o mapeamento está sendo definido.
- Uma constante definindo o modo de mapeamento. Essa constante pode assumir um dos valores a seguir.

Constante	Unidade Lógica	Val. Crescentes(X)	Val. Crescentes(Y)
MM_TEXT	Pixel	Para a Direita	Para Baixo
MM_LOMETRIC	0.1 mm	Para a Direita	Para Cima
MM_HIMETRIC	0.01 mm	Para a Direita	Para Cima
MM_LOENGLISH	0.01 pol.	Para a Direita	Para Cima
Constante	Unidade Lógica	Val. Crescentes(X)	Val. Crescentes(Y)
MM_HIENGLISH	0.001 pol.	Para a Direita	Para Cima
MM_TWIPS	1/1440 pol.	Para a Direita	Para Cima
MM_ISOTROPIC	Qualquer (x = y)	A definir	A definir
MM_ANISOTROPIC	Qualquer (x <> y)	A definir	A definir

Se quisermos, por exemplo, usar o modo de mapeamento MM_LOMETRIC para o canvas do formulário usado na aplicação descrita no tópico anterior, devemos empregar uma linha de código como a mostrada a seguir:

```
SetMapMode(FormInterativo.Canvas.Handle, MM_LOMETRIC);
```

Para converter coordenadas de dispositivo em coordenadas lógicas, deve-se utilizar a função DpToLp, que recebe como parâmetros:

- Um handle para o objeto sobre o qual o mapeamento está sendo definido.
- Uma array de variáveis do tipo TPoint.
- O número de elementos da array de variáveis do tipo TPoint.

Se você quiser transformar as coordenadas de um único ponto, não precisa definir a array – basta usar uma única variável do tipo TPoint.

O tipo TPoint é usado para representar pontos da tela, e tem dois campos, denominados x e y, que armazenam as coordenadas do ponto.

Para converter coordenadas lógicas em coordenadas de dispositivo, deve-se utilizar a função LpToDp, que recebe como parâmetros:

- Um handle para o objeto sobre o qual o mapeamento está sendo definido.
- Uma array de variáveis do tipo TPoint.
- O número de elementos da array de variáveis do tipo TPoint.

Novamente, se você quiser transformar as coordenadas de um único ponto, não precisa definir a array – basta usar uma única variável do tipo TPoint.

No caso do exemplo anterior, por exemplo, poderíamos redefinir o modo de mapeamento como MM_LOMETRIC.

Dessa maneira, o procedimento associado ao evento OnMouseMove do formulário poderia ser reescrito como mostrado a seguir:

```
procedure TFormInterativo.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
var
   pt : TPoint;
begin
     SetMapMode(FormInterativo.Canvas.Handle, MM_LOMETRIC);
     pt.x := X;
     pt.y := y;
     DPToLp(FormInterativo.Canvas.Handle,pt,1);
     Panel1.Caption := 'X = '+IntToStr(pt.X)+ ' Y = '+IntToStr(pt.y);
     If not desenha then exit;
     FormInterativo.Canvas.Pen.Color := clBlack;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
     xatual := x;
     yatual := y;
     FormInterativo.Canvas.Pen.Color := clWhite;
     FormInterativo.Canvas.MoveTo(xbase,ybase);
     FormInterativo.Canvas.LineTo(xatual,yatual);
end;
```

Observe que, nesse caso, declarou-se uma variável local ao procedimento, como mostrado no trecho de código a seguir:

```
var
   pt : TPoint;
```

Além disso, as seguintes linhas de código foram incluídas, para transformar as coordenadas de dispositivo em coordenadas lógicas, quando da apresentação dos seus valores para o usuário:

```
SetMapMode(FormInterativo.Canvas.Handle, MM_LOMETRIC);
pt.x := X;
pt.y := y;
DPToLp(FormInterativo.Canvas.Handle,pt,1);
Panel1.Caption := 'X = '+IntToStr(pt.X)+ ' Y = '+IntToStr(pt.y);
```

Os parâmetros X e Y dos procedimentos associados aos eventos OnMouseDown, OnMouseMove e OnMouseUp estão sempre definidos em termos de coordenadas de dispositivo.

Se você recompilar a aplicação e executá-la novamente, verá que os valores de Y aparecem como negativos. Isso se deve ao fato de que, nesse modo de mapeamento, o eixo Y é dirigido para cima, e mantivemos nossa origem na extremidade superior esquerda da tela.

Para alterar a posição da origem, devemos utilizar a função SetViewPortOrgEx.

Essa função é utilizada para definir o posicionamento da origem do sistema de coordenadas físicas, e recebe como parâmetros:

- Um handle para o objeto sobre o qual o mapeamento está sendo definido.
- A coordenada X, em termos de coordenadas de dispositivo, do ponto para o qual deverá se deslocar a origem.
- A coordenada Y, em termos de coordenadas de dispositivo, do ponto para o qual deverá se deslocar a origem.
- Uma variável do tipo TPoint, na qual serão armazenadas as coordenadas correntes da origem. Caso você não queira utilizar esse parâmetro, pode substituí-lo por nil.

Para alterar as dimensões a serem empregadas nas direções X e Y, no caso de se usar os modos de mapeamento MM_ISOTROPIC e MM_ANISOTROPIC, devem-se empregar as funções SetWindowExtEx e SetViewportExtEx descritas a seguir.

Funções Para Transformação de Coordenadas

As funções apresentadas a seguir permitem mapear uma área retangular (Window) em outra (ViewPort).

Função SetWindowExtEx

Essa função recebe como parâmetros:

- Um handle para o objeto sobre o qual o mapeamento está sendo definido.
- A extensão horizontal, em coordenadas lógicas, utilizada no mapeamento.
- A extensão vertical, em coordenadas lógicas, utilizada no mapeamento.

- Uma variável do tipo TPoint, que armazenará os valores correntes das extensões horizontal e vertical. Caso você não queira utilizar esse parâmetro, pode substituí-lo por nil.

Função SetViewportExtEx

Essa função recebe como parâmetros:

- Um handle para o objeto sobre o qual o mapeamento está sendo definido.
- A extensão horizontal, em coordenadas de dispositivo, utilizada no mapeamento.
- A extensão vertical, em coordenadas de dispositivo, utilizada no mapeamento.
- Uma variável do tipo TPoint, que armazenará os valores correntes das extensões horizontal e vertical. Caso você não queira utilizar esse parâmetro, pode substituí-lo por nil.

Capítulo 30

Técnicas de Impressão

Conforme descrito no capítulo anterior, uma das principais diferenças entre o sistema operacional Windows e o antigo DOS é que, enquanto este é baseado em caracteres, o primeiro constitui uma interface gráfica em que até mesmo os caracteres são desenhados. E isso se aplica não apenas à tela do monitor, mas também à superfície do papel no qual serão impressas informações de forma permanente.

Assim como ocorre com a tela, o Windows também trata uma página a ser impressa como uma superfície de desenho, isto é, um Canvas.

As funções relacionadas às tarefas de impressão do Windows também estão reunidas na biblioteca de funções denominada GDI – Graphical Device Interface. Nesse caso, a utilização dessas funções também requer, normalmente, um número que identifica o objeto sobre o qual a função vai atuar, número este denominado "handle" do objeto.

KNOW-HOW em: Impressão Direta no Windows

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos em Delphi.
- Conhecimento das principais propriedades e métodos das classes TCanvas, TBrush e TPen.

METODOLOGIA

- Apresentação do problema: Utilização das principais propriedades e métodos da classe TPrinter.

TÉCNICA

- Apresentação dos procedimentos necessários à impressão direta no Windows, utilizando as propriedades e métodos da classe TPrinter.

Existem situações em que os componentes e geradores de relatórios não atendem às necessidades específicas das nossas aplicações. Nesse caso, as rotinas de codificação devem ser feitas pelo próprio desenvolvedor.

Como para o sistema operacional Windows a superfície do papel também é considerada uma superfície de desenho, o Delphi disponibiliza uma classe denominada TPrinter, que representa a impressora configurada no sistema, e que possui uma propriedade chamada Canvas, que é um objeto da classe TCanvas, e que representa a superfície do papel no qual será feita a impressão.

Além de fornecer a classe TPrinter, o Delphi também disponibiliza um objeto dessa classe, chamado Printer, e declarado na unit Printers. Na realidade, o que a unit Printers fornece é uma função chamada Printer que retorna um objeto da classe TPrinter, mas você pode tratar Printer como um objeto da classe TPrinter, que é o tipo retornado por essa função.

Dessa maneira, para usar um objeto da classe TPrinter, basta incluir a unit Printers na cláusula uses da unit responsável pela impressão.

Para criar o desenho, basta usar os métodos da classe TCanvas, da mesma maneira que se faz em um formulário.

Nesse caso, no entanto, como a altura do papel é finita, você precisa controlar se a impressão está sendo feita dentro dos limites da página, bem como controlar o início e o encerramento dos trabalhos

de impressão. Isso será feito utilizando-se as propriedades e métodos da classe TPrinter, a serem descritas nos próximos tópicos.

Principais Propriedades da Classe TPrinter

Dentre as principais propriedades da classe TPrinter, destacam-se:

Aborted

Essa propriedade é definida como uma variável booleana, e define se um trabalho de impressão foi abortado.

Canvas

Essa propriedade é, na realidade, um objeto da classe TCanvas, sendo utilizado para representar a superfície de impressão.

Copies

Essa propriedade é declarada como uma variável inteira, e retorna o número de cópias impressas pelo dispositivo.

Fonts

Essa propriedade é definida como um objeto da classe TStrings, e retorna a lista com as fontes disponíveis para a impressora default do sistema.

Orientation

Essa propriedade, declarada como uma variável do tipo TPrinterOrientation, define a orientação da impressão na página, podendo assumir os valores poPortrait (impressão vertical) e poLandscape (impressão horizontal), valores definidos para esse tipo enumerado, como reproduzido a seguir:

```
type TPrinterOrientation = (poPortrait, poLandscape);
```

PageHeight

Essa propriedade é declarada como uma variável inteira, e define a altura da página (em pixels).

PageNumber

Essa propriedade é declarada como uma variável inteira, e retorna o número de páginas já impressas, sendo o seu valor incrementado em uma unidade a cada chamada do método NewPage da classe.

PageWidth

Essa propriedade é declarada como uma variável inteira, e define a largura da página a ser impressa (em pixels).

PrinterIndex

Essa propriedade é declarada como uma variável inteira, e retorna o índice do elemento da lista de strings definida pela propriedade Printers que corresponde à impressora selecionada no sistema operacional. Para usar a impressora default do sistema, basta atribuir o valor -1 a essa propriedade.

Printers

Essa propriedade é definida como um objeto da classe TStrings, e retorna a lista com a identificação das impressoras instaladas no sistema, sendo o índice da impressora selecionada no sistema definido pela propriedade PrinterIndex.

Conforme descrito anteriormente, quando o valor da PrinterIndex for igual a -1, será usada a impressora default do sistema.

Printing

Essa propriedade é declarada como uma variável booleana, e define se está se realizando um trabalho de impressão.

Title

Essa propriedade é declarada como uma variável do tipo string, que define o título a ser exibido no gerenciador de impressão do Windows quando o trabalho de impressão corrente estiver sendo processado.

Principais Métodos da Classe TPrinter

Dentre os principais métodos da classe TPrinter, destacam-se:

BeginDoc

Declaração

```
procedure BeginDoc;
```

Esse método deve ser chamado antes de se iniciar a impressão. Todo código de impressão deverá ser colocado entre chamadas aos métodos BeginDoc e EndDoc (a ser visto a seguir).

EndDoc

Declaração

```
procedure EndDoc;
```

Esse método deve ser chamado após a última linha do código que define um trabalho de impressão.

Na realidade, a impressão efetivamente só se inicia após uma chamada a esse método.

NewPage

Declaração

```
procedure NewPage;
```

Esse método inicia a impressão de uma nova página, e incrementa em uma unidade o valor da propriedade PageNumber.

Exemplo de Utilização da Classe TPrinter

Apresentamos a seguir um exemplo que exibe as características de cada uma das impressoras instaladas no sistema, usando-se para isso as propriedades e métodos da classe TPrinter.

Criando a Interface da Aplicação

Para criar a interface desse aplicativo-exemplo, você deve executar os seguintes procedimentos:

1. Selecione o item New Application do menu File, para iniciar o desenvolvimento de uma nova aplicação.

2. Defina os seguintes valores para as principais propriedades do formulário principal (e único) dessa aplicação-exemplo:

   Name: FormImpressoras
   Width: 470
   Height: 290
   Caption: Características das Impressoras do Sistema
   Position: poScreenCenter

3. Inclua quatro labels nesse formulário, e atribua os seguintes valores às suas principais propriedades:

   Name: LabelImpressoras
   Left: 25
   Top: 46
   Width: 162
   Height: 13
   Caption: Impressoras Instaladas no Sistema

   Name: LabelAltura
   Left: 216
   Top: 94
   Width: 142
   Height: 13
   Caption: Altura Definida Para a Página:

   Name: LabelLargura
   Left: 216

Top: 126
Width: 151
Height: 13
Caption: Largura Definida Para a Página:

Name: LabelFontes
Left: 25
Top: 94
Width: 141
Height: 13
Caption: Fontes Instaladas no Sistema:

4. Inclua um componente ListBox nesse formulário, e atribua os seguintes valores às suas principais propriedades:

   Name: ListBoxFontes
   Left: 25
   Top: 126
   Width: 176
   Height: 97
   ItemHeight: 13
   Sorted: True

5. Inclua um componente ComboBox nesse formulário, e atribua os seguintes valores às suas principais propriedades:

   Name: ComboBoxImpressoras
   Left: 216
   Top: 42
   Width: 209
   Height: 21
   ItemHeight: 13

6. Inclua um componente BitBtn nesse formulário, e atribua os seguintes valores às suas principais propriedades:

   Name: BotaoFechar
   Left: 283
   Top: 162
   Width: 75
   Height: 25
   Caption: &Fechar
   Kind: bkClose

Seu formulário deverá ficar com o aspecto apresentado na figura a seguir:

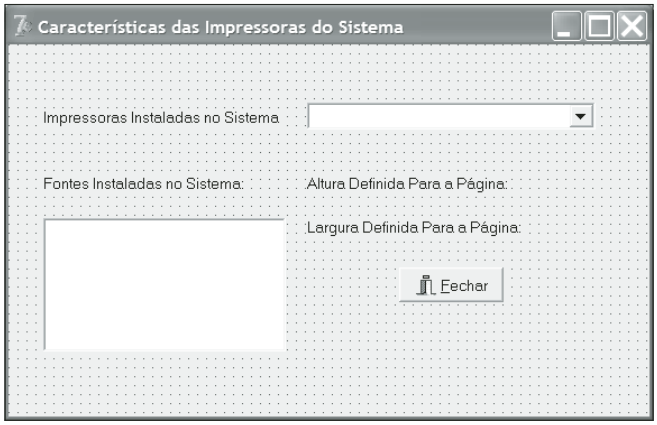


Figura 30.1: Aspecto do formulário criado para exibir as características das impressoras instaladas no sistema.

Codificando a Aplicação

Para codificar esse aplicativo-exemplo, você deve executar os seguintes procedimentos:

1. Incluir a unit Printers na cláusula Uses da unit associada a esse formulário.

2. Definir da seguinte maneira o procedimento associado ao evento OnCreate do formulário:

```
procedure TFormImpressoras.FormCreate(Sender: TObject);
begin
    ComboBoxImpressoras.Items := Printer.Printers;
    ListBoxFontes.Items := Printer.Fonts;
    LabelLargura.caption := 'Largura Definida Para a Página:
   '+IntToStr(Printer.PageWidth);
    LabelAltura.caption := 'Altura Definida Para a Página:
   '+IntToStr(Printer.PageHeight);
  ComboBoxImpressoras.ItemIndex := Printer.PrinterIndex;
end;
```

Esse procedimento atribui à propriedade Items do ComboBox (que é um objeto da classe TStrings) a lista de strings armazenada na propriedade Printers do objeto Printer, o que é feito mediante a inclusão da seguinte linha de código:

```
ComboBoxImpressoras.Items := Printer.Printers;
```

Em seguida, atribui à propriedade Items do ListBox (que é um objeto da classe TStrings) a lista de strings armazenada na propriedade Fonts do objeto Printer, o que é feito mediante a inclusão da seguinte linha de código:

```
ListBoxFontes.Items := Printer.Fonts;
```

Além disso, o valor da propriedade Caption dos Labels LabelLargura e labelAltura é atualizado para exibir, respectivamente, a largura e a altura da página definida para a impressora selecionada no sistema, o que é feito mediante a inclusão das seguintes linhas de código:

```
LabelLargura.caption := 'Largura Definida Para a Página: '+IntToStr(Printer.PageWidth);
LabelAltura.caption := 'Altura Definida Para a Página: '+IntToStr(Printer.PageHeight);
```

Por fim, atribui-se o valor da propriedade PrinterIndex do objeto Printer à propriedade ItemIndex do ComboBox, para que este exiba o nome da impressora selecionada no sistema, o que é feito mediante a inclusão das seguintes linhas de código:

```
ComboBoxImpressoras.ItemIndex := Printer.PrinterIndex;
```

3. Definir da seguinte maneira o procedimento associado ao evento OnChange do ComboBox:

```
procedure TFormImpressoras.ComboBoxImpressorasChange(Sender: TObject);
begin
    Printer.PrinterIndex := ComboBoxImpressoras.ItemIndex;
    ListBoxFontes.Items := Printer.Fonts;
    LabelLargura.caption := 'Largura Definida Para a Página:
   '+IntToStr(Printer.PageWidth);
    LabelAltura.caption := 'Altura Definida Para a Página:
   '+IntToStr(Printer.PageHeight);
end;
```

A codificação desse procedimento é muito semelhante à do procedimento anterior, razão pela qual não nos deteremos muito em comentá-las.

A única alteração, nesse caso, está no fato de que a impressora do sistema é definida de acordo com a seleção feita pelo usuário no ListBox, mediante a inclusão da seguinte linha de código:

```
Printer.PrinterIndex := ComboBoxImpressoras.ItemIndex;
```

Apresentamos a seguir a codificação completa da unit associada a esse formulário.

```
unit UnitPrinters;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TFormImpressoras = class(TForm)
    ComboBoxImpressoras: TComboBox;
    LabelImpressoras: TLabel;
    LabelAltura: TLabel;
    LabelLargura: TLabel;
    ListBoxFontes: TListBox;
    LabelFontes: TLabel;
    BotaoFechar: TBitBtn;
    procedure FormCreate(Sender: TObject);
    procedure ComboBoxImpressorasChange(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormImpressoras: TFormImpressoras;

implementation

Uses Printers;

{$R *.DFM}
```

```
procedure TFormImpressoras.FormCreate(Sender: TObject);
begin
    ComboBoxImpressoras.Items := Printer.Printers;
    ListBoxFontes.Items := Printer.Fonts;
    LabelLargura.caption := 'Largura Definida Para a Página: '+IntToStr(Printer.PageWidth);
    LabelAltura.caption := 'Altura Definida Para a Página: '+IntToStr(Printer.PageHeight);
    ComboBoxImpressoras.ItemIndex := Printer.PrinterIndex;
end;

procedure TFormImpressoras.ComboBoxImpressorasChange(Sender: TObject);
begin
    Printer.PrinterIndex := ComboBoxImpressoras.ItemIndex;
    ListBoxFontes.Items := Printer.Fonts;
    LabelLargura.caption := 'Largura Definida Para a Página: '+IntToStr(Printer.PageWidth);
    LabelAltura.caption := 'Altura Definida Para a Página: '+IntToStr(Printer.PageHeight);
end;

end.
```

Imprimindo o Conteúdo Exibido por um Componente Memo

Neste tópico, mostraremos os procedimentos necessários à impressão do conteúdo exibido por um componente Memo.

Será utilizado o exemplo criado no capítulo referente à manipulação de arquivos de texto, ao qual será acrescentada a capacidade de impressão do conteúdo exibido pelo componente Memo1.

Criando a Interface da Aplicação

Para criar a interface desse aplicativo-exemplo, você deve executar os seguintes procedimentos:

1. Selecione o item Open Project do menu File, para abrir o projeto ProjetoMemo.dpr, criado no capítulo.

2. Redefina, da seguinte maneira, o menu Arquivo do formulário principal da aplicação:

   Name: Caption
   Novo: &Novo
   Abrir: &Abrir…
   Salvar: &Salvar…
   SalvarComo: Salvar &Como…
   Separador1: -
   ConfiguraImpressora: Configurar &Impressora
   Imprimir: &Imprimir…
   Fechar: &Fechar

3. Inclua um componente PrinterSetupDialog no formulário principal da aplicação, e mantenha os valores default das suas propriedades.

Codificando a Aplicação

Para codificar esse aplicativo-exemplo, você deve executar os seguintes procedimentos:

1. Incluir a unit Printers na cláusula Uses da unit associada a esse formulário.

2. Definir da seguinte maneira o procedimento associado ao evento OnClick do item Configurar Impressora, do menu Arquivo:

```
PrinterSetupDialog1.Execute;
```

Essa linha de código faz com que a caixa de diálogo padrão para configuração de impressora do Windows seja exibida, de forma a permitir a configuração desejada.

3. Definir da seguinte maneira o procedimento associado ao evento OnClick do item Imprimir do menu Arquivo:

```
procedure TFormMemo.ImprimirClick(Sender: TObject);
var
   i, linhaatual : integer;
begin
     Printer.BeginDoc;
     linhaatual := 5*Printer.Canvas.TextHeight('A');
     for i := 0 to Memo1.Lines.Count-1 do
     begin
          if (linhaatual + trunc(1.1*Printer.Canvas.TextHeight('A'))) >=
   Printer.PageHeight
          then
             begin
                  Printer.NewPage;
                  linhaatual := 20;
             end;
          Printer.Canvas.TextOut(20, linhaatual, Memo1.Lines[i]);
          linhaatual := linhaatual + trunc(1.1*Printer.Canvas.TextHeight('A'));
     end;
     Printer.EndDoc;
end;
```

Inicialmente faz-se uma chamada ao método BeginDoc do objeto Printer, para iniciar o trabalho de impressão.

Em seguida esse procedimento define um valor para a margem superior da página, igual à altura de cinco caracteres, mediante a inclusão da seguinte linha de código, na qual o método TextHeight da propriedade Canvas do Objeto Printer é utilizado para obter a altura de um caractere (que vai depender da fonte correntemente selecionada para a impressora).

```
linhaatual := 5*Printer.Canvas.TextHeight('A');
```

Em seguida, o procedimento realiza uma iteração por todas as linhas de texto exibidas pelo componente Memo (cada uma correspondendo a uma string). Em cada iteração, verifica-se inicialmente se a distância ao topo da página é superior a um valor predefinido, o que é feito na seguinte linha de código:

```
if (linhaatual + trunc(1.1*Printer.Canvas.TextHeight('A'))) >= Printer.PageHeight then
```

Caso isso ocorra, deve-se iniciar uma nova página (o que é feito mediante uma chamada ao método NewPage do objeto Printer), reinicializando o valor da variável linhaatual.

Ao término de dada iteração do looping, imprime-se a linha corrente do componente Memo e atualiza-se o valor da variável linhaatual, o que é feito mediante a inclusão das seguintes linhas de código:

```
Printer.Canvas.TextOut(20, linhaatual, Memo1.Lines[i]);
     linhaatual := linhaatual + trunc(1.1*Printer.Canvas.TextHeight('A'));
```

Conforme pode ser verificado, a impressão é feita usando-se o método TextOut do Canvas.

Para finalizar, faz-se uma chamada ao método EndDoc do objeto Printer, para encerrar o trabalho de impressão.

Apresenta-se a seguir o código completo da unit associada ao formulário, após terem sido feitas as alterações descritas anteriormente:

```
unit UnitMemo;

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, StdCtrls, ComCtrls;
type
  TFormMemo = class(TForm)
    MainMenu1: TMainMenu;
    Arquivo: TMenuItem;
    Abrir: TMenuItem;
    Salvar: TMenuItem;
    SalvarComo: TMenuItem;
    N1: TMenuItem;
    Fechar: TMenuItem;
    Fonte1: TMenuItem;
    Negrito: TMenuItem;
    Italico: TMenuItem;
    Sublinhado: TMenuItem;
    Cortada: TMenuItem;
    N2: TMenuItem;
    SelecionarCor: TMenuItem;
    Memo1: TMemo;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    ColorDialog1: TColorDialog;
    Novo: TMenuItem;
    StatusBar1: TStatusBar;
    ConfiguraImpressora: TMenuItem;
    Imprimir: TMenuItem;
    PrinterSetupDialog1: TPrinterSetupDialog;
    procedure AbrirClick(Sender: TObject);
    procedure NovoClick(Sender: TObject);
    procedure SalvarComoClick(Sender: TObject);
    procedure SalvarClick(Sender: TObject);
    procedure FecharClick(Sender: TObject);
    procedure Memo1KeyPress(Sender: TObject; var Key: Char);
    procedure NegritoClick(Sender: TObject);
    procedure ItalicoClick(Sender: TObject);
    procedure SublinhadoClick(Sender: TObject);
    procedure CortadaClick(Sender: TObject);
    procedure SelecionarCorClick(Sender: TObject);
    procedure ImprimirClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormMemo: TFormMemo;
  Arquivoalterado: boolean;
  ArquivoAtual : string;

implementation
```

```
Uses Printers;

{$R *.DFM}

procedure TFormMemo.AbrirClick(Sender: TObject);
begin
     if OpenDialog1.Execute = true
     then
         begin
             if arquivoalterado then
             begin
                 if MessageDlg('Deseja Salvar Alterações no Arquivo Atual?',mtConfirmation,
mbOkCancel,0)= mrOk
                 then
                    Memo1.Lines.SaveToFile(arquivoatual);
             end;
             arquivoatual := OpenDialog1.FileName;
             Memo1.Lines.LoadFromFile(arquivoAtual);
             arquivoalterado := False;
             Negrito.Checked := fsbold in Memo1.Font.Style;
             Italico.Checked := fsItalic in Memo1.Font.Style;
             Cortada.Checked := fsStrikeOut in Memo1.Font.Style;
             Sublinhado.Checked := fsUnderline in Memo1.Font.Style;
             StatusBar1.Panels[0].Text := 'Nome do Arquivo: '+ arquivoatual;
         end;
end;

procedure TFormMemo.NovoClick(Sender: TObject);
begin
        if arquivoalterado then
        begin
           if MessageDlg('Deseja Salvar Alterações no Arquivo Atual?',mtConfirmation,
mbOkCancel,0)= mrOk
           then
               Memo1.Lines.SaveToFile(arquivoatual);
        end;
        arquivoatual := '';
        Memo1.Lines.Clear;
        arquivoalterado := False;
        Negrito.Checked := False;
        Italico.Checked := False;
        Cortada.Checked := False;
        Sublinhado.Checked := False;
        Memo1.Font.Style := Memo1.Font.Style - [fsBold];
        Memo1.Font.Style := Memo1.Font.Style - [fsItalic];
        Memo1.Font.Style := Memo1.Font.Style - [fsUnderline];
        Memo1.Font.Style := Memo1.Font.Style - [fsStrikeOut];
        StatusBar1.Panels[0].Text := 'Nome do Arquivo:';
end;

procedure TFormMemo.SalvarComoClick(Sender: TObject);
begin
     if SaveDialog1.Execute = true
     then
         begin
             arquivoatual := SaveDialog1.FileName;
             Memo1.Lines.SaveToFile(arquivoAtual);
             arquivoalterado := False;
             StatusBar1.Panels[0].Text := 'Nome do Arquivo: '+ arquivoatual;
         end;
end;

procedure TFormMemo.SalvarClick(Sender: TObject);
```

```
begin
     if arquivoatual = ''
     then
         SalvarComoClick(Self)
     else
         begin
             Memo1.Lines.SaveToFile(arquivoAtual);
             arquivoalterado := False;
         end;
    StatusBar1.Panels[0].Text := 'Nome do Arquivo: '+ arquivoatual;
end;

procedure TFormMemo.FecharClick(Sender: TObject);
begin
     if arquivoalterado = True
     then
         begin
             begin
                 if MessageDlg('Deseja Salvar Alterações no Arquivo Atual?',mtConfirmation,
mbOkCancel,0)= mrOk
                 then
                    if arquivoatual = ''
                    then
                        SalvarComoClick(Self)
                    else
                        Memo1.Lines.SaveToFile(arquivoatual);
             end;
         end;
         Application.Terminate;

end;

procedure TFormMemo.Memo1KeyPress(Sender: TObject; var Key: Char);
begin
     Arquivoalterado := True;
end;

procedure TFormMemo.NegritoClick(Sender: TObject);
begin
     Negrito.Checked := not Negrito.Checked;
     if Negrito.Checked
     then
         Memo1.Font.Style := Memo1.Font.Style + [fsBold]
     else
         Memo1.Font.Style := Memo1.Font.Style - [fsBold];
     arquivoalterado := True;
end;

procedure TFormMemo.ItalicoClick(Sender: TObject);
begin
     Italico.Checked := not Italico.Checked;
     if Italico.Checked
     then
         Memo1.Font.Style := Memo1.Font.Style + [fsItalic]
     else
         Memo1.Font.Style := Memo1.Font.Style - [fsItalic];
     arquivoalterado := True;
end;

procedure TFormMemo.SublinhadoClick(Sender: TObject);
begin
     Sublinhado.Checked := not Sublinhado.Checked;
     if Sublinhado.Checked
```

```
      then
          Memo1.Font.Style := Memo1.Font.Style + [fsUnderline]
      else
          Memo1.Font.Style := Memo1.Font.Style - [fsUnderline];
      arquivoalterado := True;
end;

procedure TFormMemo.CortadaClick(Sender: TObject);
begin
      Cortada.Checked := not Cortada.Checked;
      if Cortada.Checked
      then
          Memo1.Font.Style := Memo1.Font.Style + [fsStrikeOut]
      else
          Memo1.Font.Style := Memo1.Font.Style - [fsStrikeOut];
      arquivoalterado := True;
end;

procedure TFormMemo.SelecionarCorClick(Sender: TObject);
begin
      if ColorDialog1.Execute = True
      then
         begin
             Memo1.Font.Color := ColorDialog1.Color;
             arquivoalterado := True;
         end;
end;

procedure TFormMemo.ImprimirClick(Sender: TObject);
var
   i, linhaatual : integer;
begin
      Printer.BeginDoc;
      linhaatual := 5*Printer.Canvas.TextHeight('A');
      for i := 0 to Memo1.Lines.Count-1 do
      begin
           if (linhaatual + trunc(1.1*Printer.Canvas.TextHeight('A'))) >= Printer.PageHeight
           then
              begin
                   Printer.NewPage;
                   linhaatual := 20;
              end;
           Printer.Canvas.TextOut(20, linhaatual, Memo1.Lines[i]);
           linhaatual := linhaatual + trunc(1.1*Printer.Canvas.TextHeight('A'));
      end;
      Printer.EndDoc;
end;
end.
```

É claro que muitas características adicionais poderiam ser implementadas a este exemplo. O objetivo, no entanto, foi mostrar os principais procedimentos necessários à impressão direta no Windows, usando-se o objeto Printer.

Capítulo 31

Criação de DLLs

Neste capítulo serão apresentados os procedimentos necessários à criação de bibliotecas de vinculação dinâmica – DLLs – com o Borland Delphi 7.

Serão apresentados os procedimentos necessários à criação de DLLs em Delphi, bem como a sua utilização.

KNOW-HOW em: Criação de DLLs

PRÉ-REQUISITOS

- Experiência em programação orientada a objetos com a linguagem Object Pascal.

METODOLOGIA

- Apresentação do problema: Criação de uma DLL, a ser compartilhada entre diversos aplicativos.

TÉCNICA

- Apresentação dos procedimentos necessários à criação de uma DLL.

Introdução

As bibliotecas de vinculação dinâmica (Dynamic Link Libraries – DLLs) permitem que um conjunto de funções desenvolvidas em uma linguagem possa ser utilizado em programas, incluindo-se programas desenvolvidos em outras linguagens.

Você pode, por exemplo, criar uma DLL em Delphi com um conjunto de funções e utilizá-la em aplicativos desenvolvidos em C++ ou Visual Basic, por exemplo.

Procedimentos Básicos Necessários à Criação de uma DLL em Delphi

Para iniciar a criação de uma DLL, proceda da seguinte maneira:

1. Selecione o item New, do menu File, para exibir a caixa de diálogo New Items. Nessa caixa de diálogo, selecione o item DLL como mostrado na figura a seguir e depois o botão OK, para gerar o código principal da DLL.

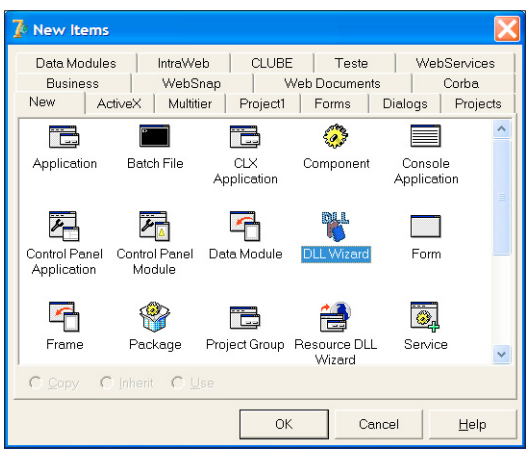


Figura 31.1: Selecionando o item DLL na caixa de diálogo New Items.

2. Será então gerado o código principal da DLL, reproduzido a seguir:

```
library Project1;

{ Important note about DLL memory management: ShareMem must be the
  first unit in your library's USES clause AND your project's (select
  Project-View Source) USES clause if your DLL exports any procedures or
  functions that pass strings as parameters or function results. This
  applies to all strings passed to and from your DLL--even those that
  are nested in records and classes. ShareMem is the interface unit to
  the BORLNDMM.DLL shared memory manager, which must be deployed along
  with your DLL. To avoid using BORLNDMM.DLL, pass string information
  using PChar or ShortString parameters. }

uses
  SysUtils,
  Classes;

{$R *.RES}

begin
end.
```

Inicialmente é exibido um comentário informando ao desenvolvedor que a unit ShareMem deve ser a primeira unit incluída na cláusula Uses do código-fonte do seu arquivo de projeto, caso a sua DLL exporte funções que recebam uma string como parâmetro ou que retornem uma string como resultado, ainda que essas strings estejam embutidas em registros ou classes. Nessa situação, o arquivo DELPHIMM.DLL deve ser distribuído junto com a sua aplicação.

Para evitar esses inconvenientes, essas strings passadas como parâmetros devem ser definidas como sendo do tipo Pchar ou ShortString (strings com no máximo 255 caracteres).

Apenas a título de exemplificação, vamos criar uma função que receba como parâmetros dois números reais e retorne o maior deles.

Essa função pode ser escrita na linguagem Object Pascal como indicado a seguir:

```
function Max(a, b : double):double;export;stdcall;
begin
     if (a > b) then result := a else result := b;
end;
```

Repare que, nesse caso, foram incluídas, no cabeçalho da função, as palavras reservadas export e stdcall.

A palavra reservada export é utilizada para indicar que essa função será exportada, isto é, poderá ser chamada por outros aplicativos.

A palavra reservada stdcall permite que os aplicativos desenvolvidos em outras linguagens façam chamadas a essa função. Isso se deve ao fato de que, no Pascal, a passagem de parâmetros é feita da esquerda para a direita, ao passo que, na maioria das outras linguagens de programação, a passagem de parâmetros é feita da direita para a esquerda.

Além disso, o nome das funções a serem exportadas deve ser incluído após a palavra reservada exports, a ser incluída após a definição das funções. Opcionalmente, poderá ser incluída a palavra reservada index, seguida por um número (índice). Dessa maneira, ao ser acessada a partir de outro aplicativo, a função poderá ser identificada pelo seu nome ou pelo seu índice.

Após fazer as alterações necessárias e salvar o projeto com o nome MaxDLL, nossa DLL ficou com o aspecto apresentado a seguir:

```
library Project1;

{ Important note about DLL memory management: ShareMem must be the
  first unit in your library's USES clause AND your project's (select
  Project-View Source) USES clause if your DLL exports any procedures or
  functions that pass strings as parameters or function results. This
  applies to all strings passed to and from your DLL—even those that
  are nested in records and classes. ShareMem is the interface unit to
  the BORLNDMM.DLL shared memory manager, which must be deployed along
  with your DLL. To avoid using BORLNDMM.DLL, pass string information
  using PChar or ShortString parameters. }

uses
  SysUtils,
  Classes;

function Max(a, b : double):double;export;stdcall;
begin
     if (a > b) then result := a else result := b;
end;

{$R *.RES}

exports Max Index 1;

begin
end.
```

Salve este projeto com o nome MaxDLL (este será o nome do arquivo no qual será armazenada a biblioteca de vinculação dinâmica – MaxDLL.dll).

Uma DLL pode ser compilada da mesma maneira que um projeto de aplicativo e, após compilada, poderá ser livremente distribuída. Você não pode, no entanto, executar uma DLL a partir do ambiente de desenvolvimento do Delphi, selecionando o item Run do menu Run (embora esse item esteja habilitado), pois uma DLL não é uma aplicação (a menos que você defina um aplicativo como Host, a ser especificado na caixa de diálogo Run Parameters, exibida quando se seleciona o item Parameters do menu Run).

Entre as palavras reservadas begin e end, que formam o corpo principal da DLL, deverá ser inserido qualquer código a ser executado durante a inicialização da DLL.

Uma DLL pode usar formulários e objetos definidos em outras units, conforme será visto posteriormente.

Chamando uma DLL a Partir de Outra Aplicação

Neste tópico será criada uma aplicação que acesse a função definida na DLL criada anteriormente.

Essa aplicação será bastante simples, composta por um único formulário no qual serão incluídas duas caixas de texto – nas quais o usuário deverá digitar dois números reais –, um botão de comando – em cujo procedimento associado ao evento OnClick será feita a chamada da função – e um label, no qual será exibido o resultado da função.

Apresentamos, a seguir, o formulário principal (e único) da aplicação e a unidade de código a ele associada.

O Formulário Principal da Aplicação

O formulário principal da aplicação é apresentado na figura a seguir:

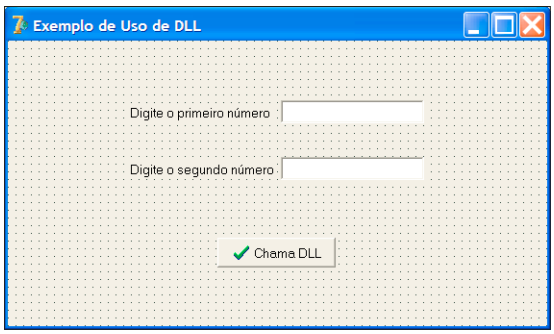


Figura 31.2: Formulário principal da aplicação.

Arquivo de Código Associado ao Formulário

Apresenta-se a seguir a codificação completa da unit associada ao formulário.

```
Arquivo UnitUsaDLL.pas
unit UnitUsaDLL;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TFormUSADLL = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Edit2: TEdit;
    BotaoChamaDLL: TBitBtn;
    procedure BotaoChamaDLLClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormUSADLL: TFormUSADLL;

implementation

{$R *.DFM}

function Max(a, b : double):double;external 'MaxDLL';

procedure TFormUSADLL.BotaoChamaDLLClick(Sender: TObject);
var
   x, y, resultado : double;
begin
```

```
    x := StrToFloat(Edit1.Text);
    y := StrToFloat(Edit2.Text);
    resultado := Max(x,y);
    ShowMessage('Valor Máximo '+FloatToStr(resultado));
end;

end.
```

Repare que a função Max é declarada na seção interface e implementada na seção implementation. Nesse caso, no entanto, informamos que, na realidade, a função está implementada em uma DLL. Como não foi informado o diretório no qual esta DLL está localizada, esta será procurada no próprio diretório da aplicação e no diretório C:\Windows\SYSTEM.

Execute a aplicação e teste a chamada da função definida na DLL.

Exibindo Formulários a Partir de uma DLL

Neste tópico serão apresentados os procedimentos necessários à exibição de uma caixa de diálogo a partir de uma chamada em uma DLL.

Será criado um formulário bastante simples, como indicado na figura a seguir, apenas para ilustrar os procedimentos adotados. Para criar este formulário, basta executar os seguintes procedimentos:

1. Selecionar o item New Aplication, do menu File, para criar uma nova aplicação.
2. Altere a propriedade Name do formulário principal desta aplicação para FormDLL.
3. Defina a propriedade Caption deste formulário como mostrado na figura a seguir (basta definir a propriedade Caption como sendo igual ao exibido na barra de títulos do formulário mostrado nesta figura).
4. Inserir um componente Label no formulário, com a propriedade Caption armazenando o texto mostrado na figura a seguir,.
5. Inserir um componente BitBtn no formulário, e definir sua propriedade Kind como bkOk.

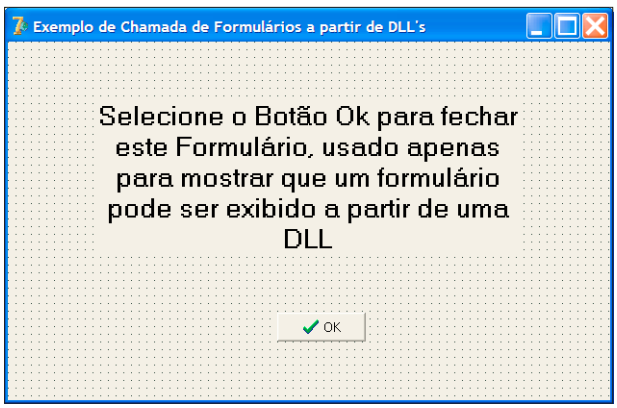


Figura 31.3: O formulário-exemplo.

O arquivo de código associado a esse formulário será bastante simples, sem a inclusão de nenhum código específico, como pode ser constatado a seguir:

```
unit UnitChamaDLL;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TFormDLL = class(TForm)
    Label1: TLabel;
    BotaoOk: TBitBtn;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormDLL: TFormDLL;

implementation

{$R *.DFM}

end.
```

Vamos incorporar essa unit à cláusula Uses da nossa DLL e definir, na DLL, o código da função MostraDLL, usada para exibir o formulário, conforme apresentado na listagem a seguir:

```
library MaxDLL;

{ Important note about DLL memory management: ShareMem must be the
  first unit in your library's USES clause AND your project's (select
  View-Project Source) USES clause if your DLL exports any procedures or
  functions that pass strings as parameters or function results. This
  applies to all strings passed to and from your DLL - even those that
  are nested in records and classes. ShareMem is the interface unit to
  the DELPHIMM.DLL shared memory manager, which must be deployed along
  with your DLL. To avoid using DELPHIMM.DLL, pass string information
  using PChar or ShortString parameters. }

uses
  SysUtils,
  Classes,
  UnitChamaDLL in 'UnitChamaDLL.pas' {FormDLL};

  function Max(a, b : double):double;export;stdcall;
  begin
     if (a > b) then result := a else result := b;
  end;

  procedure MostraDLL;export;stdcall;
  begin
        FormDLL := TFormDLL.Create(nil);
        FormDLL.ShowModal;
        FormDLL.Free;
  end;
```

```
exports
    Max index 1,
    MostraDLL index 2;

begin
end.
```

Pronto! A partir de agora, nosso formulário poderá ser exibido a partir de qualquer aplicação desenvolvida para o ambiente Windows que acesse essa DLL. Basta que se inclua, na seção implementation da unit a partir da qual será chamado este procedimento, a seguinte linha de código:

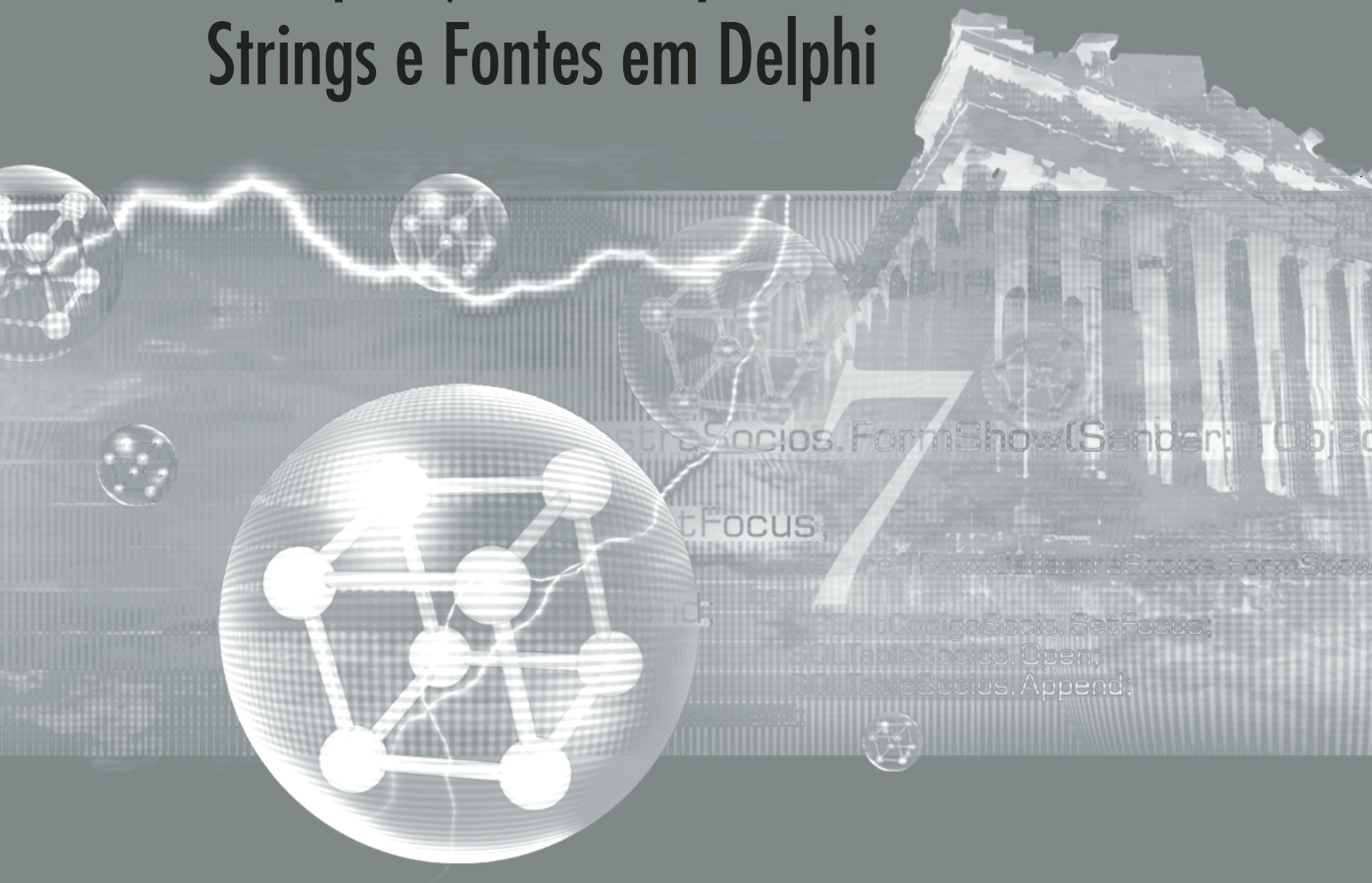
```
function MostraDLL;external 'MaxDLL';
```

Carregamento Explícito de uma DLL

Para caregar explicitamente uma DLL a partir de uma aplicação, deve-se usar a função LoadLibrary, que recebe como parâmetro o nome da DLL e cujo valor retornado deve ser armazenado em uma variável do tipo HModuke, variável esta que posteriormente será passada como parâmetro em uma chamada da função FreeLibrary, que descarregará a DLL.

Capítulo 32

Manipulação de Arquivos, Strings e Fontes em Delphi



Por mais que as linguagens de programação tenham evoluído, a maioria dos sistemas desenvolvidos na atualidade ainda necessita manipular diretamente arquivos e strings.

A fim de facilitar o trabalho do desenvolvedor, o Delphi fornece um amplo conjunto de funções e componentes que facilitam a manipulação desses tipos de dados.

Neste capítulo, serão apresentadas as principais funções, procedimentos e componentes que simplificam a manipulação de arquivos e strings em aplicações desenvolvidas em Delphi.

KNOW-HOW em: Manipulação de Arquivos

PRÉ-REQUISITOS

- Experiência em programação estruturada com versões anteriores da linguagem Pascal.
- Experiência em programação orientada a objetos em Delphi.

METODOLOGIA

- Apresentação das funções, procedimentos, classes e componentes que permitem a manipulação de arquivos em Delphi.

Manipulação Direta de Arquivos Associados a uma Variável

O Delphi permite a manipulação direta de arquivos da mesma maneira que nas antigas versões da linguagem Pascal.

Inicialmente deve-se declarar uma variável que defina o tipo de dado manipulado pelo arquivo, incluindo-se uma linha de código com a seguinte sintaxe na seção var da unit:

```
F : Tipo;
```

Onde "Tipo" pode ser:

- TextFile, para arquivos de texto ASCII.
- File Of < tipo predefinido>, para arquivos binários que armazenam dados de um determinado tipo. O tipo predefinido pode ser um tipo de variável ou uma classe.
- File, para arquivos binários que armazenem informações genéricas em formato de dado, não associadas a um tipo específico de dado.

Apresenta-se a seguir uma relação das principais funções utilizadas na manipulação direta de arquivos associados a uma variável.

Append

Declaração

Esse procedimento, declarado na unit System, abre o arquivo associado à variável F e posiciona o ponteiro de leitura/gravação no final do arquivo.

AssignFile

Para verificar se o arquivo possui um dos seguintes atributos, basta fazer uma operação lógica "and" entre o valor retornado pela função e a constante que representa cada um dos atributos.

Constante	Significado
faReadOnly	Arquivo apenas de leitura
faHidden	Arquivo Oculto
faSysFile	Arquivo de sistema
faVolumeID	Identificador de dispositivo
faDirectory	Diretório
faArchive	Arquivo comum
faAnyFile	Qualquer arquivo

FileGetDate

Declaração

```
function FileGetDate(Handle: Integer): Integer;
```

Essa função, declarada na unit SysUtils, retorna a data de um arquivo (no formato data-hora do DOS) cujo handle é passado como primeiro parâmetro na forma de um valor inteiro. Esse handle é retornado por uma chamada à função FileOpen, utilizada para abrir o arquivo.

FileOpen

Declaração

```
function FileOpen(const FileName: string; Mode: Integer): Integer;
```

Essa função, declarada na unit SysUtils, abre um arquivo cujo nome é passado como primeiro parâmetro na forma de uma string, e com o modo de abertura passado como segundo parâmetro, retornando um inteiro que pode ser usado como um handle para o arquivo.

FileSearch

Declaração

```
function FileSearch(const Name, DirList: string): string;
```

Essa função, declarada na unit SysUtils, recebe como parâmetro duas strings, sendo que a primeira deve definir o nome do arquivo a ser pesquisado e a segunda deve definir uma lista de diretórios (separados por ponto-e-vírgula) nos quais deve ser verificada a existência do arquivo.

Essa função retorna o valor True, se o arquivo for encontrado na lista de diretórios especificada como segundo parâmetro, e false, em caso contrário.

FileSetAttr

Declaração

```
function FileSetAttr(const FileName: string; Attr: Integer): Integer;
```

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string com o nome do arquivo e um valor inteiro, que definirá o novo valor do atributo (esse valor deverá ser resultado de uma operação entre as várias constantes usadas para representar os atributos do arquivo, descritas anteriormente para a função FileGetAttr, descrita anteriormente) e retorna um valor nulo se a operação não for realizada com sucesso.

FileSetDate

Declaração

```
function FileSetDate(Handle: Integer; Age: Integer): Integer;
```

Essa função, declarada na unit SysUtils, recebe como parâmetro um handle para um arquivo e um valor inteiro que representa a data a ser atribuída ao arquivo, que deve estar no formato data-hora do DOS. A função DateTimeToFileDate pode ser usada para converter um valor do tipo TDateTime no formato data-hora do DOS.

FileWrite

Declaração

```
function FileWrite(Handle: Integer; const Buffer; Count: Integer): Integer;
```

Essa função, declarada na unit SysUtils, recebe como parâmetro um handle para um arquivo e grava nesse arquivo uma quantidade igual a Count de Bytes (passada como terceiro parâmetro) armazenados em um buffer passado como segundo parâmetro.

FindClose

Declaração

```
procedure FindClose(var F: TSearchRec);
```

Esse procedimento, declarado na unit SysUtils, recebe como parâmetro uma variável do tipo TSearchRec e libera a memória alocada por uma chamada à função FindFirst.

FindFirst

Declaração

```
function FindFirst(const Path: string; Attr: Integer; var F: TSearchRec): Integer;
```

Essa função, declarada na unit SysUtils, recebe como parâmetros uma string que define um diretório, um valor inteiro que define um conjunto de atributos possíveis para um arquivo, e uma variável do tipo TSearchRec na qual serão retornadas as informações referentes ao primeiro arquivo encontrado que atenda aos requisitos especificados. O tipo TSearchRec é definido da seguinte maneira na unit SysUtils:

```
TSearchRec: record
     Time: Integer;
```

```
      Size: Integer;
      Attr: Integer;
      Name: TFileName;
      ExcludeAttr: Integer;
      FindHandle: THandle;
      FindData: TWin32FindData;
end;
```

onde TFileName é definido como um nome alternativo para uma string, como mostrado na linha de código a seguir.

```
TFileName = string;
```

FindNext

Declaração

```
function FindNext(var F: TSearchRec): Integer;
```

Essa função, declarada na unit SysUtils, retorna o próximo arquivo que atenda às condições especificadas por uma chamada à função FindFirst.

O parâmetro TSearchRec deve ser o mesmo que foi passado na chamada à função FindFirst.

ForceDirectories

Declaração

```
procedure ForceDirectories(Dir: string);
```

Esse procedimento, declarado na unit FileCtrl, cria o diretório cujo path completo é passado como parâmetro na forma de uma string (incluindo os diretórios que formam o path – caso estes não existam).

GetCurrentDir

Declaração

Essa função, declarada na unit SysUtils, retorna na forma de uma string o diretório corrente para a aplicação.

GetDir

Declaração

Esse procedimento, declarado na unit SysUtils, retorna na variável S (passada por referência) o diretório corrente do drive passado como primeiro parâmetro.

O parâmetro D pode armazenar um dos valores apresentados na tabela a seguir.

Valor	Drive
0	Default
1	A
2	B

continua

continuação

Valor	Drive
3	C
4	D

MkDir

Declaração

Esse procedimento, declarado na unit SysUtils, cria um novo diretório, cujo path completo é passado como parâmetro na forma de uma string.

RemoveDir

Declaração

Essa função, declarada na unit SysUtils, remove o diretório cujo path é passado como parâmetro na forma de uma string, desde que esse diretório esteja vazio.

Ao contrário do procedimento RmDir, apresentado a seguir, essa função retorna True se o diretório for removido (e False, em caso contrário).

RenameFile

Declaração

Esse procedimento, declarado na unit SysUtils, renomeia o arquivo cujo nome é passado como primeiro parâmetro, atribuindo-lhe o nome passado como segundo parâmetro (ambos definidos na forma de uma string).

RmDir

Declaração

Esse procedimento, declarado na unit SysUtils, remove o diretório cujo path é passado como parâmetro na forma de uma string, desde que esse diretório esteja vazio.

SetCurrentDir

Declaração

Essa função, declarada na unit SysUtils, define como diretório corrente aquele passado como parâmetro na forma de uma string, retornando True caso a operação tenha sido realizada com sucesso.

KNOW-HOW em: Manipulação de Strings

PRÉ-REQUISITOS

- Experiência em programação estruturada com versões anteriores da linguagem Pascal.
- Experiência em programação orientada a objetos em Delphi.

METODOLOGIA

- Apresentação das funções, procedimentos, classes e componentes que permitem a manipulação de strings em Delphi.

O Delphi suporta vários tipos de strings, conhecidos como:

- Caracteres de um único byte (Single-Byte Character Set – SBCS). Nesse tipo, cada byte de uma string representa um caractere.
- Caracteres de múltiplos bytes (Multiple-Byte Character Set – MBCS). Nesse tipo, alguns caracteres são representados por mais de um byte (usados principalmente em idiomas de países da Ásia).
- Caracteres Unicode. Nesse tipo, todos os caracteres são representados por dois bytes.

Em geral, as funções cujo nome começam com a palavra Ansi se referem a strings em que os caracteres são tratados como caracteres de múltiplos bytes.

Apresenta-se a seguir uma relação das principais funções utilizadas na manipulação direta de strings em Delphi.

Principais Funções Para a Manipulação de Strings

AdjustLineBreaks

Declaração

Essa função, declarada na unit SysUtils, ajusta todas as seqüências de retorno de carro (CR) / avanço de linha (LF) definidas na string passada como parâmetro, retornando uma string com os ajustes necessários (a string original não é alterada).

Qualquer retorno de carro que não é seguido por um avanço de linha, ou qualquer avanço de linha não precedido por um retorno de carro, é corrigido.

Além disso, uma seqüência LF/CR é corrigida para CR/LF.

AnsiCompareStr

Declaração

Essa função, declarada na unit SysUtils, compara duas strings passadas como parâmetros, retornando um inteiro positivo, nulo ou negativo de acordo com as seguintes regras:

Condição	Resultado
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Essa função considera o fato de os caracteres estarem em caixa alta ou caixa baixa. Em geral, na maioria dos países, a instalação local do Windows considera caracteres minúsculos precedendo os maiúsculos

(ao contrário do que ocorre com o código ASCII desses mesmos caracteres). Caso você queira considerar a ordenação feita pelo código ASCII, deve utilizar a função CompareStr.

AnsiCompareText

Declaração

Essa função, declarada na unit SysUtils, é idêntica à função AnsiCompareStr, exceto pelo fato de não considerar se as letras estão em caixa alta ou em caixa baixa.

AnsiExtractQuotedStr

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetros uma string de terminação nula, e um caractere que representa um delimitador.

Caso esse caractere delimitador não exista como primeiro caractere da string, essa função retorna uma string vazia. Em caso contrário, retorna a string sem os delimitadores inicial e final da string. Além disso, pares de delimitadores internos à string passada como parâmetro são substituídos por um único caractere delimitador na string retornada como resultado da chamada da função.

AnsiLowerCase

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string e retorna como resultado a mesma string com todos os seus caracteres em caixa baixa (minúsculos).

AnsiPos

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro duas strings e verifica se a primeira é uma substring da segunda. Em caso positivo, retorna a posição ocupada pelo primeiro caractere da substring. Em caso negativo, retorna o valor 0.

AnsiQuotedStr

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetros uma string e um caractere que representa um delimitador, acrescentando esse caractere no início e no final da string. A string assim obtida retorna como resultado da chamada da função (a string original não é alterada).

AnsiUpperCase

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string e retorna como resultado a mesma string com todos os seus caracteres em caixa alta (maiúsculos).

CompareStr

Declaração

Essa função, declarada na unit SysUtils, compara duas strings passadas como parâmetros, retornando um inteiro positivo, nulo ou negativo de acordo com as seguintes regras:

Condição	Resultado
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Essa função considera o fato de os caracteres estarem em caixa alta ou caixa baixa e considera a ordenação dos caracteres feita pelo seu código ASCII.

CompareText

Declaração

Essa função, declarada na unit SysUtils, é idêntica à função CompareStr, exceto pelo fato de não considerar se as letras estão em caixa alta ou em caixa baixa.

Concat

Declaração

Essa função, declarada na unit System, retorna uma string obtida como resultado da concatenação de duas ou mais strings passadas como parâmetros.

Copy

Declaração

Essa função, declarada na unit System, recebe como parâmetros uma string e dois números inteiros, retornando como resultado uma string que é na realidade uma substring da string passada como parâmetro. A string resultante contém "Count" caracteres, sendo o primeiro caractere aquele que ocupa, na string original, a posição definida pelo parâmetro Index.

CurrToStr

Declaração

Essa função, definida na unit SysUtils, converte uma valor que representa um valor monetário, passado como parâmetro, em uma string.

DateToStr

Declaração

Essa função, definida na unit SysUtils, converte em uma string um valor do tipo TDateTime, usado para representar uma data, e passado como parâmetro.

Delete

Declaração

Esse procedimento, declarado na unit System, recebe como parâmetros uma string e dois números inteiros, removendo da string original "Count" caracteres, sendo o primeiro caractere aquele que ocupa, na string original, a posição definida pelo parâmetro Index.

FloatToStr

Declaração

Essa função, definida na unit SysUtils, converte em uma string o número real passado como parâmetro.

Insert

Declaração

Esse procedimento, declarado na unit System, recebe como parâmetros duas strings e um número inteiro, inserindo na string passada como primeiro parâmetro a segunda string, a partir da posição definida pelo parâmetro Index.

IntToStr

Declaração

Essa função, definida na unit SysUtils, converte em uma string o número inteiro passado como parâmetro.

IsDelimiter

Declaração

Essa função, declarada na unit System, recebe como parâmetros duas strings e um número inteiro, sendo que a primeira string armazena um conjunto de caracteres a serem pesquisados na segunda string.

Se um dos caracteres especificados pela primeira string estiver presente na segunda string, na posição indicada pelo parâmetro Index, essa função retorna True (retornando false em caso contrário).

IsPathDelimiter

Declaração

Essa função, declarada na unit System, recebe como parâmetros uma string e um número inteiro, e verifica se o caractere de barra invertida está presente na segunda string, na posição indicada pelo parâmetro Index, retornando True em caso verdadeiro (e False, em caso contrário).

LastDelimiter

Declaração

Essa função, declarada na unit System, recebe como parâmetros duas strings e retorna um inteiro que define a posição, na string, de qualquer caractere que seja igual a um dos caracteres armazenados na primeira string.

Length

Declaração

Essa função, declarada na unit System, recebe como parâmetro uma string ou array e retorna como resultado o número de caracteres da string (ou de elementos da array).

LowerCase

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string e retorna como resultado a mesma string com todos os seus caracteres em caixa baixa (minúsculos).

Pos

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro duas strings e verifica se a primeira é uma substring da segunda. Em caso positivo, retorna a posição ocupada pelo primeiro caractere da substring. Em caso negativo, retorna o valor 0.

QuotedStr

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetros uma string e retorna como resultado da chamada da função uma string igual à primeira, acrescida do delimitador (') no início e no final da string (a string original não é alterada).

SetLength

Declaração

Esse procedimento, declarado na unit System, redefine o comprimento da string ou array passado como primeiro parâmetro para o valor inteiro passado como segundo parâmetro.

SetString

Declaração

Esse procedimento, declarado na unit System, redefine o comprimento e conteúdo da string passada como primeiro parâmetro. O conteúdo é definido pelo parâmetro buffer e o comprimento, pelo parâmetro len.

Str

Declaração

Esse procedimento, declarado na unit System, converte o valor X (que pode ser um número inteiro ou real) em uma string, exibindo um número de algarismos definidos pelo valor opcional Width e com o número de casas decimais definidos pelo valor opcional Decimals. O valor convertido é retornado na variável S, passada por referência.

Por exemplo, para exibir o valor de "pi" com duas casas decimais, pode-se usar um trecho de código como o apresentado a seguir.

```
Str(pi:1:2,S2);
 ShowMessage(S2);
```

StringOfChar

Declaração

Essa função, declarada na unit System, recebe como parâmetros um caractere e um número inteiro, e retorna uma string com "Count" caracteres iguais ao passado como primeiro parâmetro.

StringReplace

Declaração

Essa função, declarada na unit SysUtils, substitui na string passada como primeiro parâmetro uma ou mais ocorrências da substring OldPattern pela string NewPattern (dependendo do valor do parâmetro Flags).

O parâmetro Flags é uma variável do tipo TReplaceFlags, definida da seguinte maneira:

```
TReplaceFlags = set of (rfReplaceAll, rfIgnoreCase);
```

Se Flags contiver o valor rfReplaceAll, todas as ocorrências de OldPattern serão substituídas pela string NewPattern (se não apenas a primeira ocorrência será substituída).

Se Flags contiver o valor rfIgnoreCase, todas as ocorrências de OldPattern serão consideradas, independentemente do fato de os caracteres estarem em caixa alta ou caixa baixa.

StrToCurr

Declaração

Essa função, definida na unit SysUtils, converte em um número do tipo currency a string passada como parâmetro.

StrToDate

Declaração

Essa função, definida na unit SysUtils, converte em um número do tipo TDateTime uma string que representa uma data, passada como parâmetro.

StrToFloat

Declaração

Essa função, definida na unit SysUtils, converte em um número real a string passada como parâmetro.

StrToInt

Declaração

Essa função, definida na unit SysUtils, converte em um número inteiro a string passada como parâmetro.

StrToIntDef

Declaração

Essa função, definida na unit SysUtils, converte em um número inteiro a string passada como primeiro parâmetro. Se a conversão não puder ser realizada, será retornado o valor inteiro passado como segundo parâmetro (tratado como valor default).

StrToTime

Declaração

Essa função, definida na unit SysUtils, converte em um número do tipo TDateTime uma string que representa uma hora, passada como parâmetro.

TimeToStr

Declaração

Essa função, definida na unit SysUtils, converte em uma string um valor do tipo TDateTime, usado para representar uma hora, e passado como parâmetro.

Trim

Declaração

Essa função, declarada na unit SysUtils, retorna como resultado uma string, obtida removendo-se os espaços em branco e caracteres de controle existentes no início e no final da string passada como parâmetro (não alterando portanto a string original).

TrimLeft

Declaração

Essa função, declarada na unit SysUtils, retorna como resultado uma string, obtida removendo-se os espaços em branco e caracteres de controle existentes no início da string passada como parâmetro (não alterando portanto a string original).

TrimRight

Declaração

Essa função, declarada na unit SysUtils, retorna como resultado uma string, obtida removendo-se os espaços em branco e caracteres de controle existentes no final da string passada como parâmetro (não alterando portanto a string original).

UpperCase

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string e retorna como resultado a mesma string com todos os seus caracteres em caixa alta (maiúsculos).

Val

Declaração

Esse procedimento, declarado na unit System, converte em um valor numérico (a ser armazenado na variável passada por referência como segundo parâmetro) a string passada como primeiro parâmetro.

Caso a conversão não seja possível, a posição do caractere que impediu a conversão é armazenada na variável Code, também passada por referência.

WrapText

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetros duas strings, uma variável do tipo TSysCharSet e um número inteiro.

A primeira string define o texto que poderá ser subdividido.

A segunda string define a string que representará a subdivisão do texto – geralmente um retorno de carro/avanço de linha, representados pela string '#13#10'.

O parâmetro nBreakChars é definido como um conjunto de caracteres, separados por vírgulas e definidos entre colchetes.

O parâmetro MaxCol define o número máximo de caracteres permitidos antes da inclusão da string passada como segundo parâmetro.

Essa função insere a string passada como segundo parâmetro na última ocorrência dos caracteres definidos no terceiro parâmetro, ou na posição definida pelo quarto parâmetro.

Principais Funções Para a Manipulação de Strings de Terminação Nula

Assim como a linguagem C++, o Delphi (que é baseado na linguagem Object Pascal) também suporta strings de terminação nula (identificadas como Pchar).

Essas strings são úteis, principalmente, nas chamadas às funções da API do Windows, muitas das quais requerem como parâmetros strings de terminação nula.

AnsiStrComp

Declaração

Essa função, declarada na unit SysUtils, compara duas strings de terminação nula passadas como parâmetros, retornando um inteiro positivo, nulo ou negativo de acordo com as seguintes regras:

Condição	Resultado
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Essa função considera o fato de os caracteres estarem em caixa alta ou caixa baixa. Em geral, na maioria dos países, a instalação local do Windows considera caracteres minúsculos precedendo os maiúsculos (ao contrário do que ocorre com o código ASCII desses mesmos caracteres). Caso você queira considerar a ordenação feita pelo código ASCII, deve utilizar a função CompareStr.

AnsiStrIComp

Declaração

Essa função, declarada na unit SysUtils, compara duas strings de terminação nula passadas como parâmetros, retornando um inteiro positivo, nulo ou negativo de acordo com as regras descritas no tópico anterior, mas sem considerar se as letras estão em caixa alta ou caixa baixa.

AnsiStrLComp

Declaração

Essa função, declarada na unit SysUtils, compara duas strings de terminação nula passadas como parâmetros, considerando-se apenas os primeiros "MaxLen" bytes, retornando um inteiro positivo, nulo ou negativo de acordo com as regras descritas no tópico anterior, considerando se as letras estão em caixa alta ou caixa baixa.

AnsiStrLower

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e retorna como resultado a mesma string de terminação nula, mas com todos os seus caracteres em caixa baixa (minúsculos).

AnsiStrPos

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro duas strings de terminação e verifica se a primeira é uma substring da segunda. Em caso positivo, retorna um ponteiro com o endereço da posição ocupada pelo primeiro caractere da substring. Em caso negativo, retorna o valor nil.

AnsiStrRScan

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e um ponteiro para caractere, e retorna um ponteiro para a última ocorrência do caractere na string. Em caso negativo, retorna o valor nil.

AnsiStrScan

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e um ponteiro para caractere, e retorna um ponteiro para a primeira ocorrência do caractere na string. Em caso negativo, retorna o valor nil.

AnsiStrUpper

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e retorna como resultado a mesma string de terminação nula, mas com todos os seus caracteres em caixa alta (maiúsculos). A string original não é afetada.

StrAlloc

Declaração

Essa função, declarada na unit SysUtils, aloca memória para uma string de terminação nula capaz de armazenar um número máximo de caracteres igual a Size-1, onde Size é um número inteiro passado como parâmetro.

StrBufSize

Declaração

Essa função, declarada na unit SysUtils, retorna o número máximo de caracteres que pode ser armazenado em uma string de terminação nula, e cuja memória foi alocada por uma chamada à função StrAlloc.

StrCat

Declaração

Essa função, declarada na unit System, retorna uma string de terminação nula obtida como resultado da concatenação de duas strings de terminação nula passadas como parâmetros.

Essa função não verifica se a string de terminação nula que irá receber o resultado da chamada dessa função é capaz de armazenar todos os caracteres resultantes da concatenação. Nessas situações, deve-se dar preferência à função StrLCat, descrita adiante.

StrComp

Declaração

Essa função, declarada na unit SysUtils, compara duas strings de terminação nula passadas como parâmetros, retornando um inteiro positivo, nulo ou negativo de acordo com as seguintes regras:

Condição	Resultado
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Essa função considera o fato de os caracteres estarem em caixa alta ou caixa baixa e considera a ordenação dos caracteres feita pelo seu código ASCII.

StrCopy

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetros duas strings de terminação nula, copiando o conteúdo da segunda na primeira e retornando a primeira string (já modificada).

Essa função não verifica se a string de terminação nula que irá receber o resultado da chamada dessa função é capaz de armazenar todos os caracteres resultantes da concatenação. Nessas situações, deve-se dar preferência à função StrLCopy, descrita adiante.

StrECopy

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetros duas strings de terminação nula, copiando o conteúdo da segunda na primeira e retornando um ponteiro para o caractere de terminação da primeira string (já modificada).

Essa função não verifica se a string de terminação nula que irá receber o resultado da chamada dessa função é capaz de armazenar todos os caracteres resultantes da concatenação.

StrEnd

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e retorna um ponteiro para o caractere de terminação da string.

StrLCat

Declaração

Essa função, declarada na unit System, retorna uma string de terminação nula obtida como resultado da concatenação de duas strings de terminação nula passadas como parâmetros. Além disso, recebe um terceiro parâmetro que indica o número máximo de caracteres que pode ter a string de terminação nula resultante dessa concatenação.

StrLComp

Declaração

Essa função, declarada na unit SysUtils, compara duas strings de terminação nula passadas como parâmetros, até um máximo de MaxLen caracteres, retornando um inteiro positivo, nulo ou negativo de acordo com as regras já descritas para a função StrComp.

Essa função considera o fato de os caracteres estarem em caixa alta ou caixa baixa e considera a ordenação dos caracteres feita pelo seu código ASCII.

StrLCopy

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetros duas strings de terminação nula e um inteiro, copiando o conteúdo da segunda (limitado a MaxLen caracteres) na primeira e retornando a primeira string (já modificada).

StrLen

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e retorna o número de caracteres da string (não considerando o caractere de terminação).

StrLIComp

Declaração

Essa função, declarada na unit SysUtils, compara duas strings de terminação nula passadas como parâmetros, até um máximo de MaxLen caracteres, retornando um inteiro positivo, nulo ou negativo de acordo com as regras já descritas para a função StrComp.

Essa função não considera o fato de os caracteres estarem em caixa alta ou caixa baixa e considera a ordenação dos caracteres feita pelo seu código ASCII.

StrLower

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e retorna como resultado a mesma string de terminação nula, mas com todos os seus caracteres em caixa baixa (minúsculos).

StrMove

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetros duas strings de terminação nula e um número inteiro, copiando exatamente esse número de caracteres (passado como terceiro parâmetro) da segunda string na primeira e retornando um ponteiro para a primeira string (já modificada).

StrNew

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e retorna uma cópia dessa string (após alocar a memória necessária com uma chamada ao método StrAlloc).

StrPCopy

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetros uma string de terminação nula e uma string no estilo Pascal, retornando uma string de terminação nula obtida copiando-se os caracteres da string no estilo Pascal passada como segundo parâmetro.

StrUpper

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e retorna como resultado a mesma string de terminação nula, mas com todos os seus caracteres em caixa alta (maiúsculos).

StrRScan

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e um ponteiro para caractere, e retorna um ponteiro para a última ocorrência do caractere na string. Em caso negativo, retorna o valor nil.

StrScan

Declaração

Essa função, declarada na unit SysUtils, recebe como parâmetro uma string de terminação nula e um ponteiro para caractere, e retorna um ponteiro para a primeira ocorrência do caractere na string. Em caso negativo, retorna o valor nil.

KNOW-HOW em: Manipulação de Listas de Strings

PRÉ-REQUISITOS

- Conhecimento da sintaxe básica da linguagem Pascal e dos fundamentos da programação orientada a objetos em Delphi.

METODOLOGIA

- Apresentação da classe TStrings, que permite a manipulação de listas de strings.

TÉCNICA

- Apresentação das características principais da classe TStrings, e de componentes que implementam propriedades como objetos da classe TStrings.

A Classe TStrings

O Delphi define uma classe chamada TStrings, que define uma série de propriedades e métodos que nos torna capazes de manipular uma lista de strings e de objetos associados a essas strings.

Muitos dos métodos dessa classe são abstratos, sendo implementados nas classes dela derivadas por herança, como por exemplo a classe TStringList.

É importante salientar que a classe TStrings permite manipular não apenas uma lista de strings, mas uma lista de objetos associados a essas strings, o que a torna uma classe bastante poderosa.

Referência: A Classe TStrings

A classe TStrings, codificada na unit Classes e reproduzida a seguir, é derivada por herança da classe TPersistent e é a classe-base para todas as classes que manipulam listas de strings (e objetos a elas associados) no Delphi.

Definição da Classe TStrings

Reproduzimos a seguir o trecho de código responsável pela definição da classe TStrings, extraído da unit Classes.

```
TStrings = class(TPersistent)
  private
    FUpdateCount: Integer;
    FAdapter: IStringsAdapter;
    function GetCommaText: string;
    function GetName(Index: Integer): string;
    function GetValue(const Name: string): string;
    procedure ReadData(Reader: TReader);
    procedure SetCommaText(const Value: string);
    procedure SetStringsAdapter(const Value: IStringsAdapter);
    procedure SetValue(const Name, Value: string);
    procedure WriteData(Writer: TWriter);
  protected
    procedure DefineProperties(Filer: TFiler); override;
    procedure Error(const Msg: string; Data: Integer);
    function Get(Index: Integer): string; virtual; abstract;
    function GetCapacity: Integer; virtual;
    function GetCount: Integer; virtual; abstract;
    function GetObject(Index: Integer): TObject; virtual;
```

```
    function GetTextStr: string; virtual;
    procedure Put(Index: Integer; const S: string); virtual;
    procedure PutObject(Index: Integer; AObject: TObject); virtual;
    procedure SetCapacity(NewCapacity: Integer); virtual;
    procedure SetTextStr(const Value: string); virtual;
    procedure SetUpdateState(Updating: Boolean); virtual;
  public
    destructor Destroy; override;
    function Add(const S: string): Integer; virtual;
    function AddObject(const S: string; AObject: TObject): Integer; virtual;
    procedure Append(const S: string);
    procedure AddStrings(Strings: TStrings); virtual;
    procedure Assign(Source: TPersistent); override;
    procedure BeginUpdate;
    procedure Clear; virtual; abstract;
    procedure Delete(Index: Integer); virtual; abstract;
    procedure EndUpdate;
    function Equals(Strings: TStrings): Boolean;
    procedure Exchange(Index1, Index2: Integer); virtual;
    function GetText: PChar; virtual;
    function IndexOf(const S: string): Integer; virtual;
    function IndexOfName(const Name: string): Integer;
    function IndexOfObject(AObject: TObject): Integer;
    procedure Insert(Index: Integer; const S: string); virtual; abstract;
    procedure InsertObject(Index: Integer; const S: string;
      AObject: TObject);
    procedure LoadFromFile(const FileName: string); virtual;
    procedure LoadFromStream(Stream: TStream); virtual;
    procedure Move(CurIndex, NewIndex: Integer); virtual;
    procedure SaveToFile(const FileName: string); virtual;
    procedure SaveToStream(Stream: TStream); virtual;
    procedure SetText(Text: PChar); virtual;
    property Capacity: Integer read GetCapacity write SetCapacity;
    property CommaText: string read GetCommaText write SetCommaText;
    property Count: Integer read GetCount;
    property Names[Index: Integer]: string read GetName;
    property Objects[Index: Integer]: TObject read GetObject write PutObject;
    property Values[const Name: string]: string read GetValue write SetValue;
    property Strings[Index: Integer]: string read Get write Put; default;
    property Text: string read GetTextStr write SetTextStr;
    property StringsAdapter: IStringsAdapter read FAdapter write SetStringsAdapter;
  end;
```

Propriedades da Classe TStrings

Apresentamos a seguir a descrição das principais propriedades definidas para a classe TStrings.

Capacity

Essa propriedade retorna o número de strings que a lista é capaz de armazenar, e não o número de strings realmente armazenados pela lista, o que é definido pela sua propriedade Count (a ser vista a seguir).

Count

Essa propriedade é declarada como uma variável inteira, e retorna o número de strings armazenadas na lista e não o número de strings que podem vir a ser armazenadas pela mesma, o que é definido pela sua propriedade Capacity, descrita anteriormente.

CommaText

Essa propriedade é declarada como uma variável do tipo string, e retorna em uma única string, separadas por vírgulas, todas as strings armazenadas na lista.

```
Names
```

Uma lista de string pode armazenar, em cada string, um par de valores na forma:

```
Nome=Valor.
```

Quando isso ocorre, a propriedade Names (que é uma array) pode ser usada para retornar o valor armazenado à esquerda do sinal de igualdade para a n-ézima string da lista.

A linha de código a seguir, por exemplo, exibe o valor armazenado à esquerda do sinal de igualdade para a segunda string da lista.

```
ShowMessage('Lista.Names[1]);
```

Nesse caso, Lista é um objeto da classe TStrings.

Objects

Essa propriedade é definida como uma array de objetos da classe TObject, e retorna o objeto associado a uma string da lista, cujo índice é indicado entre colchetes.

Strings

Essa propriedade é definida como uma array de strings, e retorna a string da lista, cujo índice é indicado entre colchetes.

Text

Essa propriedade é definida como uma variável do tipo string, e retorna em uma única string as strings armazenadas na lista, separadas por caracteres de retorno de carro e de avanço de linha.

Values

Conforme descrito anteriormente, uma lista de string pode armazenar, em cada string, um par de valores na forma:

```
Nome=Valor.
```

Quando isso ocorre, a propriedade Values (que é uma array) pode ser usada para retornar o valor armazenado à direita do sinal de igualdade para a string cujo índice é referenciado pela string (e não pela sua ordem na lista de armazenamento).

A linha de código a seguir, por exemplo, exibe o valor armazenado à direita do sinal de igualdade para a string da lista que armazena, à esquerda do sinal de igualdade, o valor "Axcel".

```
ShowMessage('Lista.Values['Axcel']);
```

Nesse caso, Lista é um objeto da classe TStrings.

Métodos da Classe TStrings

Apresentamos a seguir a descrição dos principais métodos definidos para a classe TStrings.

Add

Declaração

Esse método adiciona a string passada como parâmetro ao final da lista, retornando a posição em que a string foi inserida.

AddObject

Declaração

Esse método adiciona a string passada como primeiro parâmetro e o endereço do objeto passado como segundo parâmetro ao final da lista retornando a posição em que a string e a referência ao objeto foram inseridas.

É importante destacar que o objeto (que pode ser uma instância de qualquer classe, pois todas são derivadas de TObject) já deve ter sido criado. A lista não cria strings e objetos, apenas gerencia a lista.

AddStrings

Declaração

Esse método adiciona à lista de strings gerenciada pelo objeto uma lista de strings armazenada em outro objeto da classe TStrings (bem como as referências a objetos associados às strings da lista).

Append

Declaração

Esse método adiciona a string passada como parâmetro ao final da lista, mas, ao contrário do método Add, não retorna a posição em que a string foi inserida.

Assign

Declaração

Esse método atribui o objeto passado como parâmetro ao objeto que chama o método.

Se Source é um objeto da classe TStrings, as strings e objetos referenciados pelo objeto Source passam a ser manipuladas pelo objeto que chamou o método.

Clear

Declaração

Esse método remove todas as strings e referências a objetos gerenciados pela lista.

Delete

Declaração

Esse método remove da lista a string e a referência a objeto cujo índice é passado como parâmetro.

Equals

Declaração

Esse método compara as strings armazenadas pela lista com as strings armazenadas em outra lista de strings passada como parâmetro na chamada do método.

Esse método retorna True se todas as strings forem iguais, retornando False em caso contrário.

É importante destacar que esse método compara apenas strings, e não os objetos referenciados pela lista.

Exchange

Declaração

Esse método troca as strings armazenadas nas posições passadas como parâmetros, bem como as referências a objetos a elas associadas.

IndexOf

Declaração

Esse método retorna o índice que a string passada como parâmetro ocupa na lista. Se a string não estiver armazenada na lista, será retornado o valor -1. Se a string estiver armazenada em mais de uma posição da lista, será retornado o índice da posição correspondente à primeira ocorrência da string.

IndexOfName

Declaração

Conforme descrito anteriormente, uma lista de string pode armazenar, em cada string, um par de valores na forma:

```
Nome=Valor.
```

Quando isso ocorre, esse método retorna o índice correspondente à posição ocupada pela string que exibe, à esquerda do sinal de igualdade, a string passada como parâmetro. Se essa condição não for encontrada, será retornado o valor -1.

IndexOfObject

Declaração

Esse método retorna o índice que a string da lista associada ao objeto passado como parâmetro ocupa na lista. Se a lista não armazenar nenhuma string associada ao objeto, será retornado o valor -1.

Insert

Declaração

Esse método insere na lista, na posição passada como primeiro parâmetro, a string passada como segundo parâmetro.

InsertObject

Declaração

Esse método insere, na posição passada como primeiro parâmetro, a string passada como segundo parâmetro, e associa essa string ao objeto passado como terceiro parâmetro.

LoadFromFile

Declaração

Esse método carrega a lista de strings do objeto com as strings armazenadas em um arquivo-texto cujo nome é passado como parâmetro.

Cada linha do arquivo será armazenada como uma string da lista.

LoadFromStream

Declaração

Esse método carrega a lista de strings do objeto com as strings armazenadas em um objeto da classe TStream cujo nome é passado como parâmetro.

Move

Declaração

Esse método move a string armazenada na posição da lista indicada pelo primeiro parâmetro para a posição indicada pelo segundo parâmetro.

SaveToFile

Declaração

Esse método grava a lista de strings do objeto em um arquivo-texto cujo nome é passado como parâmetro. Cada string será armazenada em uma linha do arquivo.

SaveToStream

Declaração

Esse método grava a lista de strings do objeto em um objeto da classe TStream cujo nome é passado como parâmetro.

Componentes que Definem Propriedades Como Objetos da Classe TStrings

Existem vários objetos que implementam propriedades como objetos da classe TStrings.

Dentre esses componentes podem-se destacar:

- O componente Memo, que define a sua propriedade Lines como um objeto da classe TStrings.
- O componente ListBox, que define a sua propriedade Items como um objeto da classe TStrings.
- O componente DBListBox, que define a sua propriedade Items como um objeto da classe TStrings.
- O componente ComboBox, que define a sua propriedade Items como um objeto da classe TStrings.
- O componente DBComboBox, que define a sua propriedade Items como um objeto da classe TStrings.
- O componente MainMenu, que define a sua propriedade Items como um objeto da classe TStrings.
- O componente RadioGroup, que define a sua propriedade Items como um objeto da classe TStrings.
- O componente DBRadioGroup, que define a sua propriedade Items como um objeto da classe TStrings.
- O componente RichEdit, que define a sua propriedade Lines como um objeto da classe TStrings.
- Além de outras classes e componentes.

Fica então a seguinte pergunta: como a propriedade Lines, da classe TMemo, pode utilizar o método Delete, se este é definido como abstrato?

O que ocorre é que a unit Classes implementa a classe TMemoStrings que sobrecarrega este e outros métodos, conforme indicado no seguinte trecho de código:

```
TMemoStrings = class(TStrings)
  private
    Memo: TCustomMemo;
  protected
    function Get(Index: Integer): string; override;
    function GetCount: Integer; override;
    function GetTextStr: string; override;
    procedure Put(Index: Integer; const S: string); override;
    procedure SetTextStr(const Value: string); override;
    procedure SetUpdateState(Updating: Boolean); override;
  public
    procedure Clear; override;
    procedure Delete(Index: Integer); override;
    procedure Insert(Index: Integer; const S: string); override;
  end;
```

A classe TCustomMemo, da qual a classe TMemo é derivada por herança, implementa o objeto FLines como um objeto da classe TStrings, mas ao alocar memória para esse objeto chama o método construtor da classe TMemoStrings, como mostra o código do seu método construtor, reproduzido a seguir.

```
constructor TCustomMemo.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Width := 185;
  Height := 89;
  AutoSize := False;
  DoubleBuffered := False;
  FWordWrap := True;
```

```
    FWantReturns := True;
    FLines := TMemoStrings.Create;
    TMemoStrings(FLines).Memo := Self;
  end;
```

Dessa maneira, o campo interno FLines passará a referenciar um objeto da classe TMemoStrings, na qual vários métodos abstratos da classe TStrings são sobrecarregados.

KNOW-HOW em: Manipulação de Fontes

PRÉ-REQUISITOS

- Conhecimento da sintaxe básica da linguagem Pascal e dos fundamentos da programação orientada a objetos em Delphi.

METODOLOGIA

- Apresentação da classe TFont, que permite a manipulação de fontes em aplicações desenvolvidas em Delphi.

TÉCNICA

- Apresentação das características principais da classe TFont, e de componentes que implementam propriedades como objetos da classe TFont.

Referência: a Classe TFont

A classe TFont, codificada na unit Graphics e reproduzida a seguir, é derivada por herança da classe TGraphicObject e é a classe que permite a manipulação de fontes em Delphi.

Definição da Classe TFont

Reproduzimos a seguir o trecho de código responsável pela definição da classe TFont, extraído da unit Classes.

```
TFont = class(TGraphicsObject)
  private
    FColor: TColor;
    FPixelsPerInch: Integer;
    FNotify: IChangeNotifier;
    procedure GetData(var FontData: TFontData);
    procedure SetData(const FontData: TFontData);
  protected
    procedure Changed; override;
    function GetHandle: HFont;
    function GetHeight: Integer;
    function GetName: TFontName;
    function GetPitch: TFontPitch;
    function GetSize: Integer;
    function GetStyle: TFontStyles;
    function GetCharset: TFontCharset;
    procedure SetColor(Value: TColor);
    procedure SetHandle(Value: HFont);
    procedure SetHeight(Value: Integer);
    procedure SetName(const Value: TFontName);
    procedure SetPitch(Value: TFontPitch);
    procedure SetSize(Value: Integer);
    procedure SetStyle(Value: TFontStyles);
    procedure SetCharset(Value: TFontCharset);
  public
```

```
    constructor Create;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
    property FontAdapter: IChangeNotifier read FNotify write FNotify;
    property Handle: HFont read GetHandle write SetHandle;
    property PixelsPerInch: Integer read FPixelsPerInch write FPixelsPerInch;
  published
    property Charset: TFontCharset read GetCharset write SetCharset;
    property Color: TColor read FColor write SetColor;
    property Height: Integer read GetHeight write SetHeight;
    property Name: TFontName read GetName write SetName;
    property Pitch: TFontPitch read GetPitch write SetPitch default fpDefault;
    property Size: Integer read GetSize write SetSize stored False;
    property Style: TFontStyles read GetStyle write SetStyle;
  end;
```

Propriedades da Classe TFont

Apresentamos a seguir a descrição das principais propriedades definidas para a classe TFont.

CharSet

Essa propriedade é definida como uma variável do tipo TFontCharSet, e define o conjunto de caracteres a serem utilizados pela fonte.

Color

Essa propriedade é definida como uma variável do tipo TColor, e define a cor do texto exibido pela fonte.

Handle

Essa propriedade é definida como uma variável do tipo HFont, e retorna um handle para o objeto (a ser utilizado em chamadas às funções da API do Windows).

Height

Essa propriedade é definida como uma variável do tipo inteiro, e define a altura da fonte, em pixels.

Name

Essa propriedade é definida como uma variável do tipo TFontName (que é declarado como um alias para uma string), e define o nome da fonte.

Pitch

Essa propriedade é definida como uma variável do tipo TFontPitch, e define se todos os caracteres da fonte devem ter a mesma largura (isto é, se a fonte será monoespaçada).

O tipo TFontPitch é um tipo enumerado, declarado da seguinte maneira:

```
type TFontPitch = (fpDefault, fpVariable, fpFixed);
```

onde TfontPitch pode assumir um dos seguintes valores:

- fpDefault: Os caracteres da fonte terão a sua largura default, o que dependerá da fonte escolhida (especificada na propriedade Name).
- FpFixed: Os caracteres da fonte terão largura fixa (ou seja, serão monoespaçados).
- FpVariable: Os caracteres da fonte terão largura variável.

PixelsPerInch

Essa propriedade é declarada como uma variável inteira que define o número de pixels por polegada aplicável aos caracteres da fonte.

Size

Essa propriedade é definida como uma variável do tipo inteiro, e define a altura da fonte, em pontos.

Style

Essa propriedade é definida como uma variável do tipo TFontStyles, que é um conjunto que pode armazenar os valores apresentados na tabela a seguir, e define o estilo da fonte.

```
TFontStyle = (fsBold, fsItalic, fsUnderline, fsStrikeOut);

 TFontStyles = set of TFontStyle;
```

Valor	Significado
fsBold	A fonte será exibida em negrito.
fsItalic	A fonte será exibida em itálico.
fsUnderline	A fonte será sublinhada.
fsStrikeout	A fonte será cortada por uma linha horizontal.

Exemplo de Utilização das Classes TFont e TStrings Para Manipulação de Arquivos Texto ASCII

Será apresentado a seguir um exemplo que ilustra a utilização das classes TFont, TStrings e TMemo.

Este exemplo exibirá em um componente Memo o texto a ser armazenado em um arquivo. Serão apresentados os procedimentos individuais para manipulação de arquivos e das características individuais das fontes de um componente.

Criando a Interface da Aplicação

Para criar a interface desse aplicativo-exemplo, você deve executar os seguintes procedimentos:

1. Inicie o desenvolvimento de uma nova aplicação, selecionando o item New Application do menu File.

2. Defina os seguintes valores para as principais propriedades do formulário principal da aplicação:

   Name: FormMemo
   Caption: Exemplo de Utilização do Componente Memo
   WindowState: wsMaximized

3. Inclua um componente MainMenu e defina a seguinte estrutura de menus:

   Menu Arquivo (caption &Arquivo)
   Relação de itens do Menu Arquivo

Name	Caption
Novo	&Novo
Abrir	&Abrir…
Salvar	&Salvar…
SalvarComo	Salvar &Como…
Separador1	-
Fechar	&Fechar

   Menu Fonte (caption &Fonte)

Negrito	&Negrito
Itálico	&Itálico
Sublinhado	&Sublinhado
Cortada	&Cortada
Separador2	-
SelecionarCor	Selecionar &Cor…

4. Inclua um componente StatusBar (página Win32 da paleta de componentes) no formulário e atribua os seguintes valores às suas principais propriedades:

   Name: StatusBar1
   SimplePanel: True
   SimpleText: Nome do Arquivo:
   Align: alBottom

5. Inclua um componente Memo e atribua os seguintes valores às suas principais propriedades:

   Name: Memo1
   Align: alClient
   ScrollBars: ssBoth.

6. Apague o texto exibido pela propriedade Lines do componente Memo1.

7. Inclua um componente OpenDialog (página Dialogs da paleta de componentes) no formulário e atribua os seguintes valores às suas principais propriedades:

   Name: OpenDialog1
   Filter: Arquivos Texto|*.TXT
   DefaultExt: TXT

8. Inclua um componente SaveDialog (página Dialogs da paleta de componentes) no formulário e atribua os seguintes valores às suas principais propriedades:

Name: SaveDialog1
Filter: Arquivos Texto|*.TXT
DefaultExt: TXT

9. Inclua um componente ColorDialog no formulário e atribua os seguintes valores às suas principais propriedades:

```
Name: ColorDialog1
```

Seu formulário deverá ficar com o aspecto apresentado na figura a seguir.

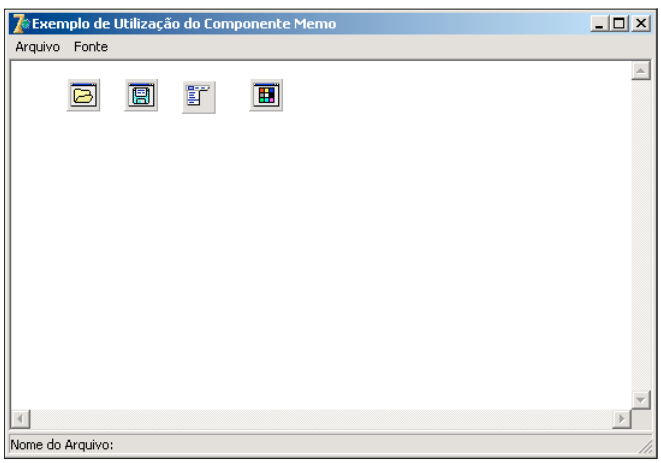


Figura 32.1: Aspecto do formulário usado no aplicativo-exemplo.

Codificando a Aplicação

Para codificar corretamente esse aplicativo, você deve executar os seguintes procedimentos:

1. Declare as seguintes variáveis globais na seção var da unit associada ao formulário:

   - Arquivoalterado: boolean;
   - ArquivoAtual: string;

A variável booleana arquivoalterado armazena o estado de alteração do arquivo.

A variável arquivoatual, do tipo string, armazena o nome do arquivo que está sendo manipulado.

2. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Novo do menu Arquivo:

```
procedure TFormMemo.NovoClick(Sender: TObject);
begin
        if arquivoalterado then
        begin
if MessageDlg('Deseja Salvar Alterações no Arquivo Atual?',Confirmation, mbOkCancel,0)=
mrOk
           then
               Memo1.Lines.SaveToFile(arquivoatual);
```

```
        end;
        arquivoatual := '';
        Memo1.Lines.Clear;
        arquivoalterado := False;
        Negrito.Checked := False;
        Italico.Checked := False;
        Cortada.Checked := False;
        Sublinhado.Checked := False;
        Memo1.Font.Style := Memo1.Font.Style - [fsBold];
        Memo1.Font.Style := Memo1.Font.Style - [fsItalic];
        Memo1.Font.Style := Memo1.Font.Style - [fsUnderline];
        Memo1.Font.Style := Memo1.Font.Style - [fsStrikeOut];
        StatusBar1.SimpleText := 'Nome do Arquivo:';
end;
```

Inicialmente esse procedimento verifica se o arquivo foi alterado após ter sido salvo pela última vez, testando o valor da variável arquivoalterado.

Caso o arquivo tenha sido alterado e não tenha sido salvo, será exibida uma mensagem perguntando ao usuário se deseja salvar as alterações efetuadas. Caso o usuário queira salvar as alterações, será executado o método SaveToFile da propriedade Lines do componente Memo1.

Por fim, os valores default da fonte do componente Memo e dos diversos itens de menu são restabelecidos, e uma string nula é atribuída à variável arquivoatual.

3. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Abrir do menu Arquivo:

```
procedure TFormMemo.AbrirClick(Sender: TObject);
begin
     if OpenDialog1.Execute = true
     then
         begin
             if arquivoalterado then
             begin
                 if MessageDlg('Deseja Salvar Alterações no Arquivo
Atual?',mtConfirmation, mbOkCancel,0)= mrOk
                 then
                    Memo1.Lines.SaveToFile(arquivoatual);
             end;
             arquivoatual := OpenDialog1.FileName;
             Memo1.Lines.LoadFromFile(arquivoAtual);
             arquivoalterado := False;
             Negrito.Checked := fsbold in Memo1.Font.Style;
             Italico.Checked := fsItalic in Memo1.Font.Style;
             Cortada.Checked := fsStrikeOut in Memo1.Font.Style;
             Sublinhado.Checked := fsUnderline in Memo1.Font.Style;
             StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
         end;
end;
```

Esse procedimento exibe a caixa de diálogo padrão para abertura de arquivos, mediante uma chamada ao método Execute do componente OpenDialog1. Caso o usuário selecione o botão Abrir dessa caixa de diálogo, o método Execute retornará o valor True e o arquivo selecionado (armazenado na propriedade FileName do componente OpenDialog) é armazenado na variável arquivoatual.

O arquivo é carregado usando-se o método LoadFromFile da propriedade Lines do componente Memo (propriedade esta que é, conforme descrito anteriormente, um objeto da classe TStrings) e atribui-se o

valor false à variável arquivo alterado (pois o arquivo acaba de ser carregado – e portanto nenhuma alteração foi feita).

Repare ainda que, caso o arquivo anteriormente manipulado tenha sido alterado e essas alterações não tenham sido salvas, solicita-se uma confirmação por parte do usuário.

Por fim, o valor da propriedade Checked de cada um dos diversos itens do menu Fonte é definida em função dos elementos armazenados na propriedade Style da propriedade Font (que é um objeto da classe TFont) do componente Memo1. Além disso, o nome do arquivo atual é atribuído à propriedade SimpleText do componente StatusBar.

4. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Salvar Como do menu Arquivo:

```
procedure TFormMemo.SalvarComoClick(Sender: TObject);
begin
     if SaveDialog1.Execute = true
     then
         begin
             arquivoatual := SaveDialog1.FileName;
             Memo1.Lines.SaveToFile(arquivoAtual);
             arquivoalterado := False;
             StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
         end;
end;
```

Esse procedimento exibe a caixa de diálogo padrão para gravação de arquivos, mediante uma chamada ao método Execute do componente SaveDialog1. Caso o usuário selecione o botão Salvar dessa caixa de diálogo, o método Execute retornará o valor True e o arquivo selecionado (armazenado na propriedade FileName do componente SaveDialog) será armazenado na variável arquivoatual.

O arquivo é gravado usando-se o método SaveToFile da propriedade Lines do componente Memo (propriedade esta que é, conforme descrito anteriormente, um objeto da classe TStrings) e atribui-se o valor false à variável arquivoalterado (pois o arquivo acaba de ser salvo). Além disso, o nome do arquivo atual é atribuído à propriedade SimpleText do componente StatusBar.

5. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Salvar do menu Arquivo:

```
procedure TFormMemo.SalvarClick(Sender: TObject);
begin
     if arquivoatual = ''
     then
         SalvarComoClick(Self)
     else
         begin
             Memo1.Lines.SaveToFile(arquivoAtual);
             arquivoalterado := False;
         end;
    StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
end;
```

Esse procedimento verifica, inicialmente, se ainda não foi atribuído um nome para o arquivo e, em caso positivo, executa o procedimento associado ao evento OnClick do item Salvar Como do menu Arquivo (repare a utilização da propriedade Self).

Caso o arquivo já tenha recebido um nome, as últimas alterações são salvas mediante uma chamada ao método SaveToFile da propriedade Lines do componente Memo. Conseqüentemente, atribui-se o valor false à variável arquivoalterado.

Por fim, o nome do arquivo é atribuído à propriedade SimpleText do componente StatusBar.

6. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Fechar do menu Arquivo:

```
procedure TFormMemo.FecharClick(Sender: TObject);
begin
     if arquivoalterado = True
     then
         begin
             begin
                 if MessageDlg('Deseja Salvar Alterações no Arquivo
Atual?',mtConfirmation, mbOkCancel,0)= mrOk
                 then
                    if arquivoatual = ''
                    then
                        SalvarComoClick(Self)
                    else
                        Memo1.Lines.SaveToFile(arquivoatual);
             end;
         end;
         Application.Terminate;
end;
```

Esse procedimento verifica, inicialmente, se existem alterações a serem gravadas e, em caso positivo, pergunta ao usuário se deseja salvar as alterações. Em caso positivo esse procedimento verifica se ainda não foi atribuído um nome para o arquivo e, em caso positivo, executa o procedimento associado ao evento OnClick do item Salvar Como do menu Arquivo (se não, executa o método SaveToFile da propriedade Lines do componente Memo1).

Por fim, a aplicação é encerrada, mediante a execução do método Terminate do objeto Application, da classe TApplication (que representa a aplicação).

7. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Negrito do menu Fonte:

```
procedure TFormMemo.NegritoClick(Sender: TObject);
begin
     Negrito.Checked := not Negrito.Checked;
     if Negrito.Checked
     then
         Memo1.Font.Style := Memo1.Font.Style + [fsBold]
     else
         Memo1.Font.Style := Memo1.Font.Style - [fsBold];
     arquivoalterado := True;
end;
```

Esse método muda o valor da propriedade Checked do item de Menu.

Caso o valor dessa propriedade passe a ter o valor True, o valor da subpropriedade Style da propriedade Font do componente Memo1 passa a incluir o elemento fsBold no seu conjunto de valores.

Caso o valor dessa propriedade passe a ter o valor False, o valor da subpropriedade Style da propriedade Font do componente Memo1 exclui o elemento fsBold do seu conjunto de valores.

8. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Itálico do menu Fonte:

```
procedure TFormMemo.ItalicoClick(Sender: TObject);
begin
     Italico.Checked := not Italico.Checked;
     if Italico.Checked
     then
         Memo1.Font.Style := Memo1.Font.Style + [fsItalic]
     else
         Memo1.Font.Style := Memo1.Font.Style - [fsItalic];
     arquivoalterado := True;
end;
```

Esse método muda o valor da propriedade Checked do item de menu correspondente.

Caso o valor dessa propriedade passe a ter o valor True, o valor da subpropriedade Style da propriedade Font do componente Memo1 passa a incluir o elemento fsItalic no seu conjunto de valores.

Caso o valor dessa propriedade passe a ter o valor False, o valor da subpropriedade Style da propriedade Font do componente Memo1 exclui o elemento fsItalic do seu conjunto de valores.

9. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Sublinhado do menu Fonte:

```
procedure TFormMemo.SublinhadoClick(Sender: TObject);
begin
     Sublinhado.Checked := not Sublinhado.Checked;
     if Sublinhado.Checked
     then
         Memo1.Font.Style := Memo1.Font.Style + [fsUnderline]
     else
         Memo1.Font.Style := Memo1.Font.Style - [fsUnderline];
     arquivoalterado := True;
end;
```

Esse método muda o valor da propriedade Checked do item de Menu.

Caso o valor dessa propriedade passe a ter o valor True, o valor da subpropriedade Style da propriedade Font do componente Memo1 passa a incluir o elemento fsUnderline no seu conjunto de valores.

Caso o valor dessa propriedade passe a ter o valor False, o valor da subpropriedade Style da propriedade Font do componente Memo1 exclui o elemento fsUnderline do seu conjunto de valores.

10. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Cortada do menu Fonte:

```
procedure TFormMemo.CortadaClick(Sender: TObject);
begin
     Cortada.Checked := not Cortada.Checked;
     if Cortada.Checked
     then
         Memo1.Font.Style := Memo1.Font.Style + [fsStrikeOut]
     else
```

```
            Memo1.Font.Style := Memo1.Font.Style - [fsStrikeOut];
        arquivoalterado := True;
    end;
```

Esse método muda o valor da propriedade Checked do item de menu correspondente.

Caso o valor dessa propriedade passe a ter o valor True, o valor da subpropriedade Style da propriedade Font do componente Memo1 passa a incluir o elemento fsStrikeOut no seu conjunto de valores.

Caso o valor dessa propriedade passe a ter o valor False, o valor da subpropriedade Style da propriedade Font do componente Memo1 exclui o elemento fsStrikeOut do seu conjunto de valores.

11. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Selecionar Cor do menu Fonte:

```
procedure TFormMemo.SelecionarCorClick(Sender: TObject);
begin
     if ColorDialog1.Execute = True
     then
        begin
            Memo1.Font.Color := ColorDialog1.Color;
            arquivoalterado := True;
        end;
end;
```

Esse método exibe a caixa de diálogo padrão para seleção de cores do Windows, mediante uma chamada ao método Execute do componente ColorDialog1.

Caso o usuário, após selecionar a cor desejada, selecione o botão Ok dessa caixa de diálogo, a mesma será fechada e retornará o valor True. A cor selecionada (e armazenada na propriedade Color do componente ColorDialog) será atribuída à subpropriedade Color da propriedade Font do componente Memo1. Além disso, o valor True será atribuído à variável arquivoalterado.

12. Defina da seguinte maneira o procedimento associado ao evento OnKeyPress do componente Memo1:

```
procedure TFormMemo.Memo1KeyPress(Sender: TObject; var Key: Char);
begin
     Arquivoalterado := True;
end;
```

Esse procedimento apenas atribui o valor True à propriedade arquivoalterado, indicando que o arquivo sofreu modificação.

Apresenta-se a seguir o código completo da unit associada ao formulário.

```
unit UnitMemo;
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, StdCtrls, ComCtrls;

type
  TFormMemo = class(TForm)
    MainMenu1: TMainMenu;
    Arquivo: TMenuItem;
    Abrir: TMenuItem;
```

```
    Salvar: TMenuItem;
    SalvarComo: TMenuItem;
    Separador1: TMenuItem;
    Fechar: TMenuItem;
    Fonte1: TMenuItem;
    Negrito: TMenuItem;
    Itálico: TMenuItem;
    Sublinhado: TMenuItem;
    Cortada: TMenuItem;
    Separador2: TMenuItem;
    SelecionarCor: TMenuItem;
    Memo1: TMemo;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    ColorDialog1: TColorDialog;
    Novo: TMenuItem;
    StatusBar1: TStatusBar;
    procedure AbrirClick(Sender: TObject);
    procedure NovoClick(Sender: TObject);
    procedure SalvarComoClick(Sender: TObject);
    procedure SalvarClick(Sender: TObject);
    procedure FecharClick(Sender: TObject);
    procedure Memo1KeyPress(Sender: TObject; var Key: Char);
    procedure NegritoClick(Sender: TObject);
    procedure ItalicoClick(Sender: TObject);
    procedure SublinhadoClick(Sender: TObject);
    procedure CortadaClick(Sender: TObject);
    procedure SelecionarCorClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormMemo: TFormMemo;
  Arquivoalterado: boolean;
  ArquivoAtual : string;

implementation

{$R *.DFM}

procedure TFormMemo.AbrirClick(Sender: TObject);
begin
     if OpenDialog1.Execute = true
     then
         begin
             if arquivoalterado then
             begin
                 if MessageDlg('Deseja Salvar Alterações no Arquivo Atual?',mtConfirmation,
mbOkCancel,0)= mrOk
                 then
                    Memo1.Lines.SaveToFile(arquivoatual);
             end;
             arquivoatual := OpenDialog1.FileName;
             Memo1.Lines.LoadFromFile(arquivoAtual);
             arquivoalterado := False;
             Negrito.Checked := fsbold in Memo1.Font.Style;
             Italico.Checked := fsItalic in Memo1.Font.Style;
             Cortada.Checked := fsStrikeOut in Memo1.Font.Style;
             Sublinhado.Checked := fsUnderline in Memo1.Font.Style;
             StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
         end;
```

```
end;

procedure TFormMemo.NovoClick(Sender: TObject);
begin
        if arquivoalterado then
        begin
           if MessageDlg('Deseja Salvar Alterações no Arquivo Atual?',mtConfirmation,
mbOkCancel,0)= mrOk
           then
               Memo1.Lines.SaveToFile(arquivoatual);
        end;
        arquivoatual := '';
        Memo1.Lines.Clear;
        arquivoalterado := False;
        Negrito.Checked := False;
        Italico.Checked := False;
        Cortada.Checked := False;
        Sublinhado.Checked := False;
        Memo1.Font.Style := Memo1.Font.Style - [fsBold];
        Memo1.Font.Style := Memo1.Font.Style - [fsItalic];
        Memo1.Font.Style := Memo1.Font.Style - [fsUnderline];
        Memo1.Font.Style := Memo1.Font.Style - [fsStrikeOut];
        StatusBar1.SimpleText := 'Nome do Arquivo:';
end;

procedure TFormMemo.SalvarComoClick(Sender: TObject);
begin
     if SaveDialog1.Execute = true
     then
         begin
             arquivoatual := SaveDialog1.FileName;
             Memo1.Lines.SaveToFile(arquivoAtual);
             arquivoalterado := False;
             StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
         end;
end;

procedure TFormMemo.SalvarClick(Sender: TObject);
begin
     if arquivoatual = ''
     then
         SalvarComoClick(Self)
     else
         begin
             Memo1.Lines.SaveToFile(arquivoAtual);
             arquivoalterado := False;
         end;
    StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
end;

procedure TFormMemo.FecharClick(Sender: TObject);
begin
     if arquivoalterado = True
     then
         begin
             begin
                 if MessageDlg('Deseja Salvar Alterações no Arquivo Atual?',mtConfirmation,
mbOkCancel,0)= mrOk
                 then
                    if arquivoatual = ''
                    then
                        SalvarComoClick(Self)
                    else
                        Memo1.Lines.SaveToFile(arquivoatual);
```

```
            end;
         end;
         Application.Terminate;
end;

procedure TFormMemo.Memo1KeyPress(Sender: TObject; var Key: Char);
begin
     Arquivoalterado := True;
end;

procedure TFormMemo.NegritoClick(Sender: TObject);
begin
     Negrito.Checked := not Negrito.Checked;
     if Negrito.Checked
     then
         Memo1.Font.Style := Memo1.Font.Style + [fsBold]
     else
         Memo1.Font.Style := Memo1.Font.Style - [fsBold];
     arquivoalterado := True;
end;

procedure TFormMemo.ItalicoClick(Sender: TObject);
begin
     Italico.Checked := not Italico.Checked;
     if Italico.Checked
     then
         Memo1.Font.Style := Memo1.Font.Style + [fsItalic]
     else
         Memo1.Font.Style := Memo1.Font.Style - [fsItalic];
     arquivoalterado := True;
end;

procedure TFormMemo.SublinhadoClick(Sender: TObject);
begin
     Sublinhado.Checked := not Sublinhado.Checked;
     if Sublinhado.Checked
     then
         Memo1.Font.Style := Memo1.Font.Style + [fsUnderline]
     else
         Memo1.Font.Style := Memo1.Font.Style - [fsUnderline];
     arquivoalterado := True;
end;

procedure TFormMemo.CortadaClick(Sender: TObject);
begin
     Cortada.Checked := not Cortada.Checked;
     if Cortada.Checked
     then
         Memo1.Font.Style := Memo1.Font.Style + [fsStrikeOut]
     else
         Memo1.Font.Style := Memo1.Font.Style - [fsStrikeOut];
     arquivoalterado := True;
end;

procedure TFormMemo.SelecionarCorClick(Sender: TObject);
begin
     if ColorDialog1.Execute = True
     then
        begin
            Memo1.Font.Color := ColorDialog1.Color;
            arquivoalterado := True;
        end;
end;

end.
```

Exemplo de Utilização das Classes TFont e TStrings Para Manipulação de Arquivos RTF

Será apresentado a seguir um exemplo que ilustra a utilização das classes TFont e TStrings, além de apresentar algumas características do componente RichEdit.

Esse exemplo exibirá em um componente RichEdit o texto a ser armazenado em um arquivo RTF. Serão apresentados os procedimentos individuais para manipulação de arquivos e das características individuais das fontes de um componente.

Criando a Interface da Aplicação

Para criar a interface desse aplicativo-exemplo, você deve executar os seguintes procedimentos:

1. Inicie o desenvolvimento de uma nova aplicação, selecionando o item New Application do menu File.
2. Defina os seguintes valores para as principais propriedades do formulário principal da aplicação:

   Name: FormRTF
   Caption: Exemplo de Utilização do Componente RichEdit
   WindowState: wsMaximized

3. Inclua um componente MainMenu e defina a seguinte estrutura de menus:

Menu Arquivo (caption &Arquivo)
- Relação de itens do Menu Arquivo

Name	Caption
Novo	&Novo
Abrir	&Abrir…
Salvar	&Salvar…
SalvarComo	Salvar &Como…
Separador1	-
Fechar	&Fechar

Menu Fonte (caption &Fonte)

Selecionar	&Selecionar
Separador2	-
SelecionarCor	Selecionar &Cor…

4. Inclua um componente StatusBar no formulário e atribua os seguintes valores às suas principais propriedades:

   Name: StatusBar1
   SimplePanel: True
   SimpleText: Nome do Arquivo:
   Align: alBottom

5. Inclua um componente RichEdit e atribua os seguintes valores às suas principais propriedades:

Name: RichEdit1
Align: alClient
ScrollBars: ssBoth.

6. Apague o texto exibido pela propriedade Lines do componente RichEdit1.
7. Inclua um componente OpenDialog no formulário e atribua os seguintes valores às suas principais propriedades:

   Name: OpenDialog1
   Filter: Arquivos RTF|*.RTF
   DefaultExt: RTF

8. Inclua um componente SaveDialog no formulário e atribua os seguintes valores às suas principais propriedades:

   Name: SaveDialog1
   Filter: Arquivos RTF|*.RTF
   DefaultExt: RTF

9. Inclua um componente ColorDialog no formulário e atribua os seguintes valores às suas principais propriedades:

   ```
   Name: ColorDialog1
   ```

10. Inclua um componente FontDialog no formulário e atribua os seguintes valores às suas principais propriedades:

    ```
    Name: FontDialog1
    ```

Seu formulário deverá ficar com o aspecto apresentado na figura a seguir.

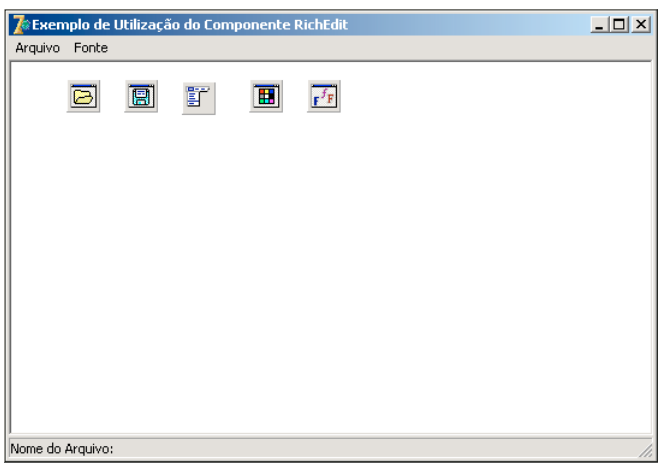


Figura 32.2: Aspecto do formulário usado no aplicativo-exemplo.

Codificando a Aplicação

Para codificar corretamente esse aplicativo, você deve executar os seguintes procedimentos:

1. Declare as seguintes variáveis globais na seção var unit associada ao formulário:

```
Arquivoalterado: boolean;
ArquivoAtual : string;
```

A variável booleana arquivoalterado armazena o estado de alteração do arquivo.

A variável arquivoatual, do tipo string, armazena o nome do arquivo que está sendo manipulado.

2. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Novo do menu Arquivo:

```
procedure TFormRTF.NovoClick(Sender: TObject);
begin
        if arquivoalterado then
        begin
           if MessageDlg('Deseja Salvar Alterações no Arquivo Atual?',mtConfirmation,
mbOkCancel,0)= mrOk
           then
               RichEdit1.Lines.SaveToFile(arquivoatual);
        end;
        arquivoatual := '';
        RichEdit1.Lines.Clear;
        arquivoalterado := False;
        RichEdit1.Font.Style := RichEdit1.Font.Style - [fsBold];
        RichEdit1.Font.Style := RichEdit1.Font.Style - [fsItalic];
        RichEdit1.Font.Style := RichEdit1.Font.Style - [fsUnderline];
        RichEdit1.Font.Style := RichEdit1.Font.Style - [fsStrikeOut];
        StatusBar1.SimpleText := 'Nome do Arquivo:';
end;
```

Inicialmente esse procedimento verifica se o arquivo foi alterado após ter sido salvo pela última vez, testando o valor da variável arquivoalterado.

Caso o arquivo tenha sido alterado e não tenha sido salvo, será exibida uma mensagem perguntando ao usuário se deseja salvar as alterações efetuadas. Caso o usuário queira salvar as alterações, será executado o método SaveToFile do componente RichEdit1.

Por fim, os valores default da fonte do componente RichEdit são restabelecidos, e uma string nula é atribuída à variável arquivoatual.

3. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Abrir do menu Arquivo:

```
procedure TFormRTF.AbrirClick(Sender: TObject);
begin
     if OpenDialog1.Execute = true
     then
         begin
             if arquivoalterado then
             begin
                 if MessageDlg('Deseja Salvar Alterações no Arquivo
Atual?',mtConfirmation, mbOkCancel,0)= mrOk
                 then
```

```
                RichEdit1.Lines.SaveToFile(arquivoatual);
            end;
            arquivoatual := OpenDialog1.FileName;
            RichEdit1.Lines.LoadFromFile(arquivoAtual);
            arquivoalterado := False;
            StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
        end;
end;
```

Esse procedimento exibe a caixa de diálogo padrão para abertura de arquivos, mediante uma chamada ao método Execute do componente OpenDialog1. Caso o usuário selecione o botão Abrir dessa caixa de diálogo, o método Execute retornará o valor True e o arquivo selecionado (armazenado na propriedade FileName do componente OpenDialog) é armazenado na variável arquivoatual.

O arquivo é carregado usando-se o método LoadFromFile da propriedade Lines do componente RichEdit (propriedade esta que é, conforme descrito anteriormente, um objeto da classe TStrings) e atribui-se o valor False à variável arquivoalterado (pois o arquivo acaba de ser carregado – e portanto nenhuma alteração foi feita).

Repare ainda que, caso o arquivo tenha sido alterado e essas alterações não tenham sido salvas, solicita-se uma confirmação por parte do usuário.

4. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Salvar Como do menu Arquivo:

```
procedure TFormRTF.SalvarComoClick(Sender: TObject);
begin
     if SaveDialog1.Execute = true
     then
         begin
             arquivoatual := SaveDialog1.FileName;
             RichEdit1.Lines.SaveToFile(arquivoAtual);
             arquivoalterado := False;
             StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
         end;
end;
```

Esse procedimento exibe a caixa de diálogo padrão para gravação de arquivos, mediante uma chamada ao método Execute do componente SaveDialog1. Caso o usuário selecione o botão Salvar dessa caixa de diálogo, o método Execute retornará o valor True e o arquivo selecionado (armazenado na propriedade FileName do componente SaveDialog) é armazenado na variável arquivoatual.

O arquivo é gravado usando-se o método SaveToFile da propriedade Lines do componente RichEdit (propriedade esta que é, conforme descrito anteriormente, um objeto da classe TStrings) e atribui-se o valor False à variável arquivoalterado (pois o arquivo acaba de ser salvo – e portanto nenhuma alteração foi feita). Além disso, o nome do arquivo atual é atribuído à propriedade SimpleText do componente StatusBar.

5. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Salvar do menu Arquivo:

```
procedure TFormRTF.SalvarClick(Sender: TObject);
begin
     if arquivoatual = ''
     then
         SalvarComoClick(Self)
```

```
        else
            begin
                RichEdit1.Lines.SaveToFile(arquivoAtual);
                arquivoalterado := False;
            end;
        StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
end;
```

Esse procedimento verifica, inicialmente, se ainda não foi atribuído um nome para o arquivo e, em caso positivo, executa o procedimento associado ao evento OnClick do item Salvar Como do menu Arquivo.

Caso o arquivo já tenha recebido um nome, as últimas alterações são salvas mediante uma chamada ao método SaveToFile da propriedade Lines do componente RichEdit. Conseqüentemente, atribui-se o valor False à variável arquivoalterado.

Por fim, o nome do arquivo é atribuído à propriedade SimpleText do componente StatusBar.

6. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Fechar do menu Arquivo:

```
procedure TFormRTF.FecharClick(Sender: TObject);
begin
     if arquivoalterado = True
     then
         begin
             begin
                 if MessageDlg('Deseja Salvar Alterações no Arquivo
Atual?',mtConfirmation, mbOkCancel,0)= mrOk
                 then
                    if arquivoatual = ''
                    then
                        SalvarComoClick(Self)
                    else
                        RichEdit1.Lines.SaveToFile(arquivoatual);
             end;
         end;
         Application.Terminate;
end;
```

Esse procedimento verifica, inicialmente, se existem alterações a serem gravadas e, em caso positivo, pergunta ao usuário se deseja salvar as alterações. Em caso positivo esse procedimento verifica se ainda não foi atribuído um nome para o arquivo e, em caso positivo, executa o procedimento associado ao evento OnClick do item Salvar Como do menu Arquivo (se não, executa o método SaveToFile da propriedade Lines do componente RichEdit1).

Por fim, a aplicação é encerrada, mediante a execução do método Terminate do objeto Application, da classe TApplication.

7. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Selecionar do menu Fonte:

```
procedure TFormRTF.SelecionarClick(Sender: TObject);
begin
     if RichEdit1.SelLength > 0
     then
```

```
        begin
            if FontDialog1.Execute = True
            then
            begin
               RichEdit1.SelAttributes.Assign(FontDialog1.Font)
               arquivoalterado := True;
            end;
        end;
end;
```

Inicialmente esse procedimento verifica se existe algum texto selecionado (analisando o valor da propriedade SelLength do componente RichEdit1) e, em caso positivo, exibe a caixa de diálogo de seleção de fonte, mediante uma chamada ao método Execute do componente FontDialog1.

Caso o usuário selecione o botão Ok para fechar essa caixa de diálogo, a fonte escolhida nessa caixa de diálogo será atribuída ao texto selecionado do componente RichEdit, mediante a execução da seguinte linha de código:

RichEdit1.SelAttributes.Assign(FontDialog1.Font)

Por fim, atribui-se o valor True à propriedade arquivoalterado, indicando que foram feitas alterações, e que estas ainda não foram salvas.

8. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Selecionar Cor do menu Fonte:

```
procedure TFormRTF.SelecionarCorClick(Sender: TObject);
begin
     if ColorDialog1.Execute = True
     then
        begin
            RichEdit1.SelAttributes.Color := ColorDialog1.Color;
            arquivoalterado := True;
        end;
end;
```

Esse método exibe a caixa de diálogo padrão para seleção de cores, mediante uma chamada ao método Execute do componente ColorDialog1.

Caso o usuário, após selecionar a cor desejada, selecione o botão Ok dessa caixa de diálogo, a mesma será fechada e retornará o valor True. A cor selecionada (e armazenada na propriedade Color do componente ColorDialog) será atribuída à subpropriedade Color do texto selecionado no componente RichEdit1. Além disso, o valor True será atribuído à variável arquivoalterado.

9. Defina da seguinte maneira o procedimento associado ao evento OnKeyPress do componente RichEdit1:

```
procedure TFormRTF.RichEdit1KeyPress(Sender: TObject; var Key: Char);
begin
     Arquivoalterado := True;
end;
```

Esse procedimento apenas atribui o valor True à propriedade arquivoalterado, indicando que o arquivo sofreu modificação.

Apresenta-se a seguir o código completo da unit associada ao formulário.

```
unit UnitRichEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, StdCtrls, ComCtrls;

type
  TFormRTF = class(TForm)
    MainMenu1: TMainMenu;
    Arquivo: TMenuItem;
    Abrir: TMenuItem;
    Salvar: TMenuItem;
    SalvarComo: TMenuItem;
    Separador1: TMenuItem;
    Fechar: TMenuItem;
    Fonte1: TMenuItem;
    Selecionar: TMenuItem;
    Separador2: TMenuItem;
    SelecionarCor: TMenuItem;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    ColorDialog1: TColorDialog;
    Novo: TMenuItem;
    StatusBar1: TStatusBar;
    RichEdit1: TRichEdit;
    FontDialog1: TFontDialog;
    procedure AbrirClick(Sender: TObject);
    procedure NovoClick(Sender: TObject);
    procedure SalvarComoClick(Sender: TObject);
    procedure SalvarClick(Sender: TObject);
    procedure FecharClick(Sender: TObject);
    procedure RichEdit1KeyPress(Sender: TObject; var Key: Char);
    procedure SelecionarClick(Sender: TObject);
    procedure SelecionarCorClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormRTF: TFormRTF;
  Arquivoalterado: boolean;
  ArquivoAtual : string;

implementation

{$R *.DFM}

procedure TFormRTF.AbrirClick(Sender: TObject);
begin
     if OpenDialog1.Execute = true
     then
         begin
             if arquivoalterado then
             begin
                 if MessageDlg('Deseja Salvar Alterações no Arquivo
Atual?',mtConfirmation, mbOkCancel,0)= mrOk
                 then
```

```
                    RichEdit1.Lines.SaveToFile(arquivoatual);
                end;
                arquivoatual := OpenDialog1.FileName;
                RichEdit1.Lines.LoadFromFile(arquivoAtual);
                arquivoalterado := False;
                StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
            end;
end;

procedure TFormRTF.NovoClick(Sender: TObject);
begin
        if arquivoalterado then
        begin
           if MessageDlg('Deseja Salvar Alterações no Arquivo Atual?',mtConfirmation,
mbOkCancel,0)= mrOk
           then
               RichEdit1.Lines.SaveToFile(arquivoatual);
        end;
        arquivoatual := '';
        RichEdit1.Lines.Clear;
        arquivoalterado := False;
        RichEdit1.Font.Style := RichEdit1.Font.Style - [fsBold];
        RichEdit1.Font.Style := RichEdit1.Font.Style - [fsItalic];
        RichEdit1.Font.Style := RichEdit1.Font.Style - [fsUnderline];
        RichEdit1.Font.Style := RichEdit1.Font.Style - [fsStrikeOut];
        StatusBar1.SimpleText := 'Nome do Arquivo:';
end;

procedure TFormRTF.SalvarComoClick(Sender: TObject);
begin
     if SaveDialog1.Execute = true
     then
         begin
             arquivoatual := SaveDialog1.FileName;
             RichEdit1.Lines.SaveToFile(arquivoAtual);
             arquivoalterado := False;
             StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
         end;
end;

procedure TFormRTF.SalvarClick(Sender: TObject);
begin
     if arquivoatual = ''
     then
         SalvarComoClick(Self)
     else
         begin
             RichEdit1.Lines.SaveToFile(arquivoAtual);
             arquivoalterado := False;
         end;
     StatusBar1.SimpleText := 'Nome do Arquivo: '+ arquivoatual;
end;

procedure TFormRTF.FecharClick(Sender: TObject);
begin
     if arquivoalterado = True
     then
         begin
             begin
                 if MessageDlg('Deseja Salvar Alterações no Arquivo
Atual?',mtConfirmation, mbOkCancel,0)= mrOk
                 then
                    if arquivoatual = ''
                    then
```

```
                            SalvarComoClick(Self)
                        else
                            RichEdit1.Lines.SaveToFile(arquivoatual);
                    end;
                end;
                Application.Terminate;
        end;

        procedure TFormRTF.RichEdit1KeyPress(Sender: TObject; var Key: Char);
        begin
             Arquivoalterado := True;
        end;

        procedure TFormRTF.SelecionarClick(Sender: TObject);
        begin
             if RichEdit1.SelLength > 0
        then
           begin
               if FontDialog1.Execute = True
               then
               begin
                  RichEdit1.SelAttributes.Assign(FontDialog1.Font)
                  arquivoalterado := True;
               end;
           end;
    end;

    procedure TFormRTF.SelecionarCorClick(Sender: TObject);
    begin
         if ColorDialog1.Execute = True
         then
           begin
               RichEdit1.SelAttributes.Color := ColorDialog1.Color;
               arquivoalterado := True;
           end;
    end;

    end.
```

Capítulo 33

Manipulação de Threads em Delphi

Neste capítulo apresentaremos o conceito de Threads, que permitem que vários processos sejam executados "simultaneamente" em uma aplicação escrita em Delphi.

KNOW-HOW em: Threads

PRÉ-REQUISITOS

- Fundamentos da programação orientada a objetos em Delphi.
- Utilização do Ambiente de Desenvolvimento Integrado do Delphi 7.

METODOLOGIA

- Apresentação do conceito de Threads e processos.

TÉCNICA

- Apresentação dos procedimentos necessários à utilização de Threads em uma aplicação desenvolvida em Delphi.

O Conceito de Threads

Existem situações em que pode ser desejável que vários processos sejam executados "simultaneamente" pelo processador, isto é, que uma tarefa não deva esperar o processamento completo de outra tarefa para poder iniciar a sua execução.

A palavra simultaneamente foi colocada entre aspas porque, exceto no caso de processamento paralelo (múltiplos processadores em uma mesma máquina ou em um "cluster"), essa simultaneidade real não é possível.

Nada impede, no entanto, que o processador possa dedicar uma parte do seu tempo a cada uma de várias tarefas (processos), de forma que para o usuário esse processamento "pareça simultâneo".

Consideremos inicialmente a situação descrita a seguir.

Uma aplicação é composta por um único formulário no qual são inseridos três componentes ProgressBar e dois botões de comando. Acrescenta-se manualmente à classe do formulário um método chamado Progresso, que incrementa o valor da propriedade Position de um componente ProgressBar (passado como parâmetro na chamada ao método) a partir de um valor inicial definido pela sua propriedade Min, até atingir o valor máximo definido pela sua propriedade Max.

O aspecto desse formulário e a unidade de código associada são apresentados na Figura 33.1.

Para que os efeitos destes exemplos possam ser visualizados com maior nitidez, altere a propriedade Max de cada componente ProgressBar para 10000.

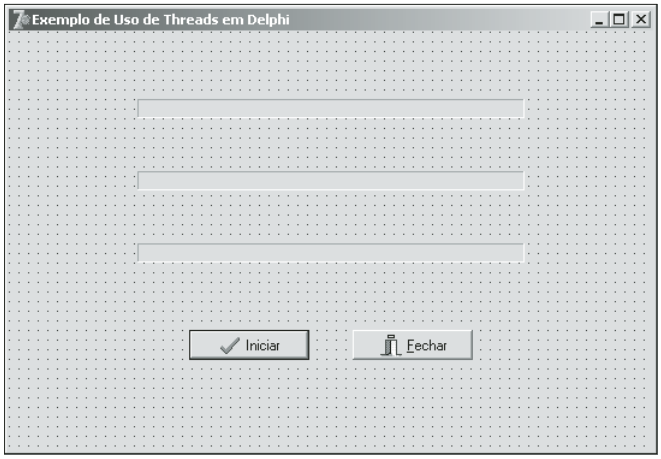


Figura 33.1: Aspecto do formulário.

Unidade de Código Associada:

```
unit UnitPBThread;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ComCtrls;

type
  TFormThread = class(TForm)
    ProgressBar1: TProgressBar;
    ProgressBar2: TProgressBar;
    ProgressBar3: TProgressBar;
    BotaoIniciar: TBitBtn;
    BotaoFechar: TBitBtn;
    procedure BotaoIniciarClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure Progresso(PB : TProgressBar);
  end;

var
  FormThread: TFormThread;

implementation

{$R *.DFM}

{ TFormThread }

procedure TFormThread.Progresso(PB: TProgressBar);
begin
  While (PB.Position < PB.Max) do
  PB.Position := PB.Position+1;
end;

procedure TFormThread.BotaoIniciarClick(Sender: TObject);
begin
```

```
      ProgressBar1.Position := 0;
      ProgressBar2.Position := 0;
      ProgressBar3.Position := 0;
      Progresso(Progressbar1);
      Progresso(Progressbar2);
      Progresso(Progressbar3);
  end;

  end.
```

Ao executar essa aplicação e selecionar o botão Iniciar, verifica-se que cada componente ProgressBar atualiza o valor da sua propriedade Position de forma independente. Os valores da segunda ProgressBar só começam a ser atualizados após o valor da primeira atingir o seu valor máximo, e os valores da terceira ProgressBar só começam a ser atualizados após o valor da segunda atingir o seu valor máximo.

Evidentemente, isso se deve ao fato de que o processo representado pelo método "Progresso" é executado uma vez para cada componente, e cada novo processo só se inicia após o processo anterior ter sido concluído.

A figura a seguir apresenta a execução do segundo processo (após a conclusão do primeiro), sendo efetuado antes de se iniciar a execução do terceiro processo.

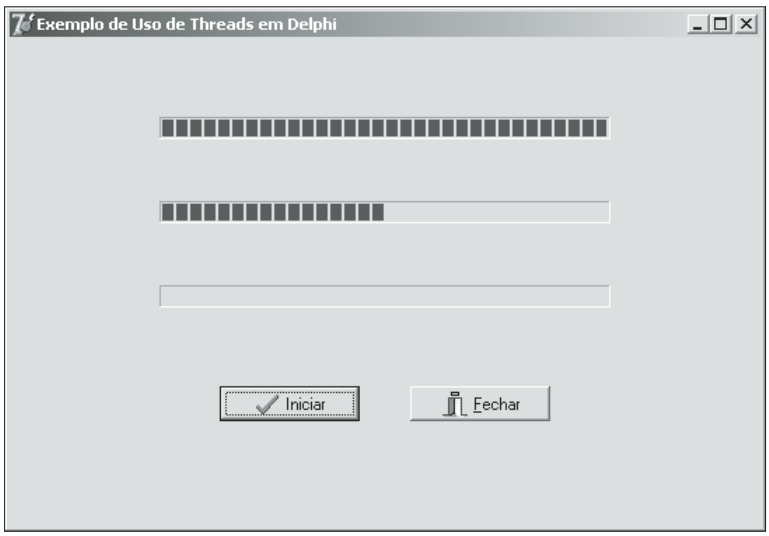


Figura 33.2: Execução independente dos diversos processos.

Para permitir a execução de vários processos "simultaneamente", o Delphi utiliza o conceito de Threads, implementado através da classe TThread.

A classe TThread, definida na unit Classes, é a classe-base para todos os processos a serem implementados e manipulados pela aplicação, sendo descrita no próximo tópico.

A Classe TThread

Conforme descrito anteriormente, cada processo a ser executado em uma aplicação pode ser representado por um objeto de uma classe derivada da classe TThread.

A necessidade de se derivar uma classe a partir da classe TThread se deve ao fato de que o método Execute, responsável pela execução do processo, é definido como um método abstrato na classe TThread. Conseqüentemente, para criar um processo, você deve inicialmente criar uma classe derivada da classe TThread e obrigatoriamente implementar o seu método Execute.

Para facilitar a criação de novas classes derivadas de TThread você pode executar os seguintes procedimentos:

1. Selecionar o item New/Other do menu File do Delphi 7 para exibir a caixa de diálogo New Items.

2. Selecionar o item Thread Object da página New dessa caixa de diálogo e o botão Ok, para exibir a caixa de diálogo New Thread Object mostrada na Figura 33.3, na qual deve ser digitado o nome da classe que representará o novo processo.

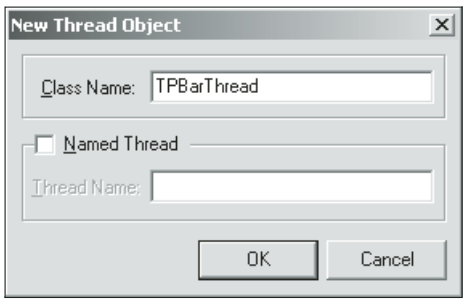


Figura 33.3: A caixa de diálogo New Thread Object.

Nesta versão do Delphi você pode dar um nome a uma Thread de modo a identificá-la durante a depuração de um programa que executa múltiplas Threads. Neste caso, marque a caixa de verificação Named Thread existente na caixa de diálogo New Thread Object, e defina o nome da Thread na caixa de texto Named Thread. Neste caso, será ainda criado um outro método, chamado SetName, na classe da Thread, e que será o primeiro método a ser disparado quando a Thread estiver sendo executada.

3. Selecione o botão Ok para fechar essa caixa de diálogo. Será gerado o esqueleto dessa nova classe, como mostrado a seguir.

```
unit UnitPBarThread;

   interface

   uses
     Classes;

   type
     TPBarThread = class(TThread)
     private
       { Private declarations }
     protected
       procedure Execute; override;
     end;

   implementation

   { Important: Methods and properties of objects in VCL can only be used in a
     method called using Synchronize, for example,
```

```
        Synchronize(UpdateCaption);

    and UpdateCaption could look like,

      procedure TPBarThread.UpdateCaption;
      begin
        Form1.Caption := 'Updated in a thread';
      end; }

{ TPBarThread }

procedure TPBarThread.Execute;
begin
  { Place thread code here }
end;

end.
```

Repare que a definição do método Execute é tão importante ela já foi gerada automaticamente, juntamente com um comentário que indica onde deve ser definido o código da thread.

Ainda aparece comentado um trecho em que se informa que alterações de propriedades e chamadas de métodos de objetos da VCL só podem ser feitas a partir de uma chamada ao método Synchronize (implementado na classe-base). Dessa maneira, qualquer método que altere uma propriedade de um componente da VCL deve ser executado passando-se o seu nome como parâmetro na chamada ao método Sinchronize da Thread.

O método construtor da classe TThread, por sua vez, tem um parâmetro que indica se a execução do processo representado pelo objeto deve ser iniciada assim que o objeto for criado. Para iniciar imediatamente o processo, você deve passar o valor False na chamada do construtor.

Caso o objeto seja criado passando-se como parâmetro o valor True, você deve reiniciar posteriormente a execução do processo mediante uma chamada ao método Resume da classe TThread. Por outro lado, para suspender posteriormente a execução do processo, basta executar o método Suspend da classe (os métodos Resume e Suspend não têm parâmetros).

Para finalizar a qualquer tempo a execução de um processo, você deve usar o método Terminate da thread (que atribui o valor True à propriedade Terminated do objeto que representa o processo). Nesse caso, se a propriedade FreeOnTerminated do objeto que representa o processo for definida como True, a memória alocada para o objeto pelo seu construtor será liberada (caso o valor da propriedade FreeOnTerminated seja igual a False, deve-se chamar explicitamente o seu método Free).

O método Execute de uma thread deve verificar periodicamente o valor da propriedade Terminated do objeto que representa a Thread, e finalizar a sua execução caso o valor dessa propriedade seja igual a True.

Você pode definir diferentes níveis de prioridade para cada processo, definindo-se adequadamente o valor da propriedade Priority do objeto que o representa. Essa propriedade pode assumir os valores descritos na tabela a seguir.

A utilização dessa propriedade é útil quando você quer, por exemplo, que o sistema execute um backup de dados enquanto o usuário executa outras tarefas mais importantes.

Valor	Significado
TpIdle	Indica uma prioridade de uma Thread que só será executada se o sistema entrar em um estado Idle (repouso).
TpLowest	Indica uma prioridade de dois pontos abaixo do normal.
TpLower	Indica uma prioridade de um ponto abaixo do normal.
TpNormal	Indica uma prioridade igual à dada à Thread principal da aplicação (normal).
TpHigher	Indica uma prioridade de um ponto acima do normal.
TpHighest	Indica uma prioridade de dois pontos acima do normal.
TpTimeCritical	Mais alta prioridade.

Implementando a Classe TPBThread

Apresenta-se a seguir o código da unit que implementa a thread que será utilizada no nosso exemplo.

```
unit UnitPBarThread;

interface

uses
  Classes, ComCtrls;

type
  TPBarThread = class(TThread)
  private
    { Private declarations }
    FPB: TProgressBar;
  protected
    procedure Execute; override;
    procedure Progresso;
  public
    constructor Cria(PB: TProgressBar);
  end;

implementation

{ Important: Methods and properties of objects in VCL can only be used in a
  method called using Synchronize, for example,

      Synchronize(UpdateCaption);

  and UpdateCaption could look like,

    procedure TPBarThread.UpdateCaption;
    begin
      Form1.Caption := 'Updated in a thread';
    end; }

{ TPBarThread }

constructor TPBarThread.Cria(PB: TProgressBar);
begin
    Create(False);
    FPB := PB;
    FreeOnTerminate := True;
end;
```

```
procedure TPBarThread.Execute;
begin
  { Place thread code here }
  While (FPB.Position < FPB.Max) do Synchronize(Progresso);
end;

procedure TPBarThread.Progresso;
begin
    FPB.Position := FPB.Position + 1;
end;

end.
```

Repare que a classe tem como propriedade um objeto da classe TProgressBar (razão pela qual a unit ComCttls foi incluída na cláusula uses da unit que declara a classe da TThread).

Observe que o objeto manipulado pela Thread é passado como parâmetro na chamada do método construtor Cria da classe, e que este chama o método Create – construtor da classe-base – com o valor False, para que a execução do processo seja iniciada automaticamente.

Atribui-se ainda o valor True à propriedade FreeOnTerminated do objeto, para que a memória alocada seja liberada automaticamente ao se terminar a execução do processo.

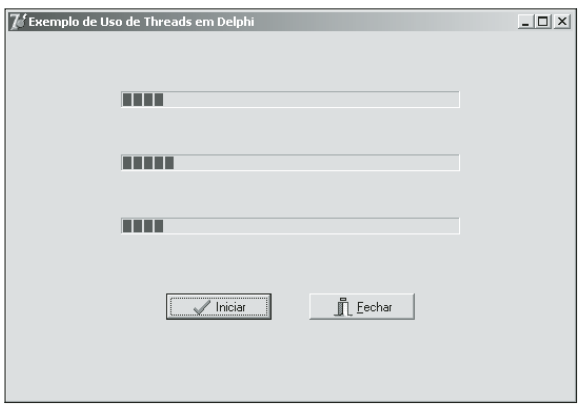
Redefinindo o Código da Unit Associada ao Formulário

Apresenta-se a seguir o código da unit associada ao formulário do nosso exemplo.

```
unit UnitPBThread;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ComCtrls;

type
  TFormThread = class(TForm)
    ProgressBar1: TProgressBar;
    ProgressBar2: TProgressBar;
    ProgressBar3: TProgressBar;
    BotaoIniciar: TBitBtn;
    Botaofechar: TBitBtn;
    procedure BotaoIniciarClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
end;

var
  FormThread: TFormThread;

implementation

uses UnitPBarThread;

{$R *.DFM}

procedure TFormThread.BotaoIniciarClick(Sender: TObject);
```

```
var
   PBarThread1, PBarThread2, PBarThread3 : TPBarThread;
begin
    ProgressBar1.Position := 0;
    ProgressBar2.Position := 0;
    ProgressBar3.Position := 0;
    PBarThread1 := TPBarThread.Cria(Progressbar1);
    PBarThread2 := TPBarThread.Cria(Progressbar2);
    PBarThread3 := TPBarThread.Cria(Progressbar3);
end;

end.
```

Repare que, no procedimento associado ao evento OnClick do botão Iniciar, são declarados três objetos da classe TPBarThread, instanciados mediante uma chamada ao método Cria da classe.

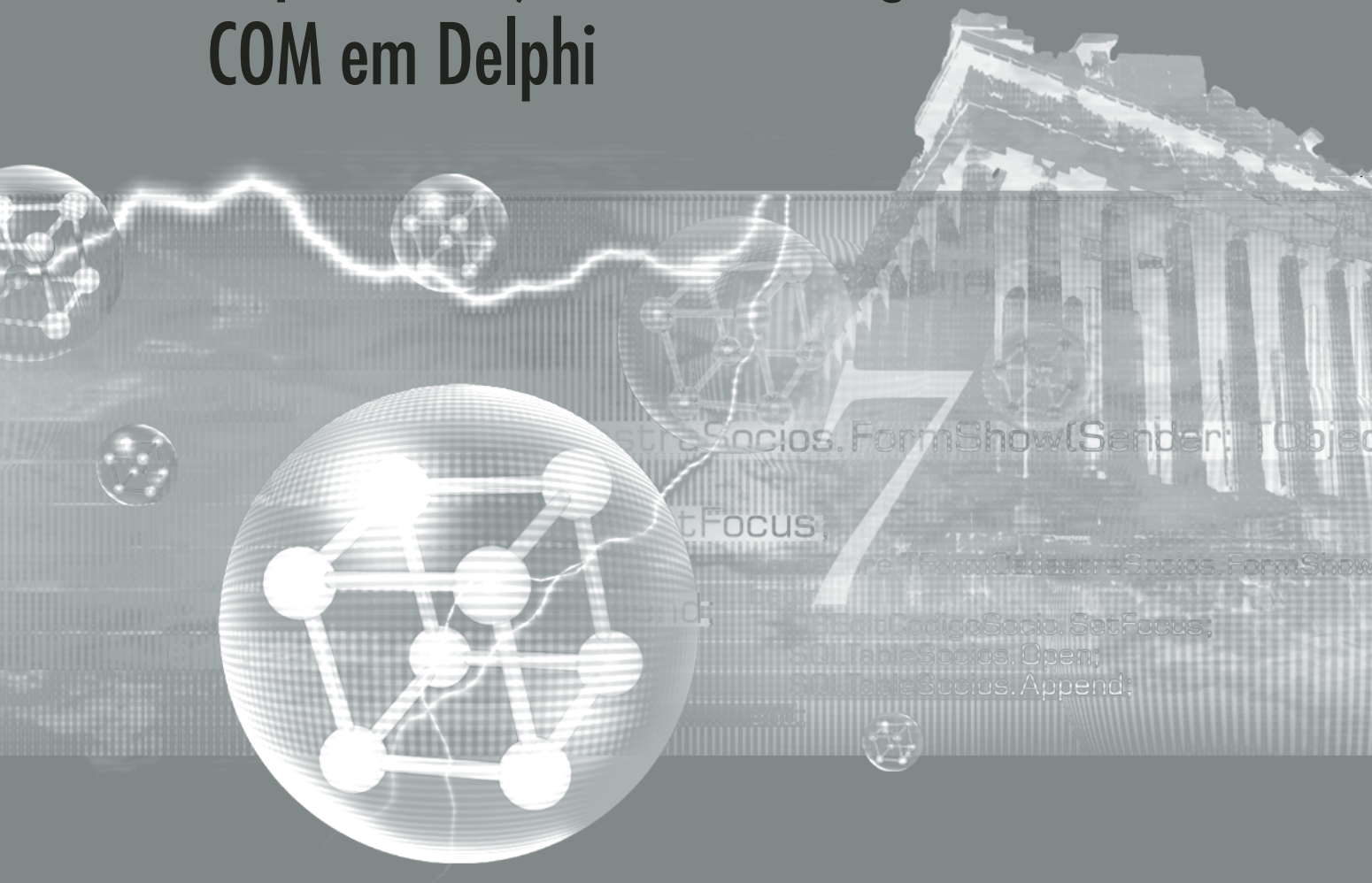


A Figura 33.4 apresenta a execução simultânea dos processos.

Evidentemente o exemplo utilizado neste capítulo é bastante simples, e poderia ter sido criado utilizando-se outras técnicas. O objetivo deste exemplo, no entanto, foi apresentar de forma clara e didática os conceitos básicos relacionados à execução "simultânea" de vários processos. Na prática, esses processos podem ser muito mais complexos, e sua solução só se torna possível mediante a utilização de threads.

Capítulo 34

Implementação da Tecnologia COM em Delphi



Neste capítulo, apresentaremos a tecnologia COM (Component Object Model) e como seus conceitos são implementados em Delphi. Esta tecnologia está presente apenas no ambiente Windows, razão pela qual só se aplica a projetos baseados na VCL.

KNOW-HOW em: Tecnologia COM

PRÉ-REQUISITOS

- Fundamentos da linguagem Object Pascal.
- Fundamentos da programação orientada a objetos em Delphi.
- Utilização do ambiente de desenvolvimento integrado do Delphi.

METODOLOGIA

- Apresentação dos conceitos relacionados à tecnologia COM, e sua implementação em Delphi.

TÉCNICA

- Apresentação dos procedimentos necessários à criação de objetos COM em Delphi.

A Tecnologia COM

A tecnologia COM (na qual estão baseadas as tecnologias OLE e ActiveX) compreende uma especificação definida pela Microsoft para a criação de componentes reutilizáveis de software.

Como essa especificação define componentes em um formato binário, sua implementação e utilização podem, a princípio, ser feitas em qualquer linguagem e sistema operacional.

Assim como um objeto da VCL, um objeto COM pode apresentar uma série de funcionalidades, disponibilizadas através das suas interfaces (interfaces são semelhantes a classes, pois reúnem em uma única entidade várias funções e procedimentos).

Uma interface, conforme descrito anteriormente, pode declarar vários métodos. Entretanto, é importante salientar que uma interface apenas declara os métodos – sem implementá-los. Conseqüentemente, uma interface é muito semelhante a uma classe que declara apenas métodos abstratos.

A implementação desses métodos é deixada a cargo de uma classe denominada CoClass, que pode implementar uma ou mais interfaces (devendo, no entanto, implementar todos os métodos de cada interface).

Além disso, diferentemente de uma classe – que é identificada por um nome – uma interface é identificada por um número, denominado GUID (Global Unique Identifier), que identifica univocamente uma interface (ao menos estatisticamente).

Esse número é definido em Delphi como um tipo composto, declarado da seguinte maneira na unit System.pas:

```
TGUID = record
    D1: LongWord;
    D2: Word;
    D3: Word;
    D4: array[0..7] of Byte;
  end;
```

Você não precisa, no entanto, se preocupar em gerar esse número, pois o Delphi cuidará dessa tarefa para você sempre que se criar um novo objeto COM.

Se, por alguma razão, você precisar gerarexplicitamente um GUID, basta pressionar simultaneamente as teclas Ctrl, Shift e G no editor de códigos do Delphi.

Um objeto COM deve ser instanciado sempre que uma aplicação precisar utilizar seus serviços (implementados através das suas interfaces), e pode ser utilizado por várias aplicações simultaneamente. Além disso, quando não estiver sendo mais utilizado, deve ser capaz de se autodestruir (e liberar a memória utilizada).

Para que isso seja possível, a especificação COM determina que cada objeto deve ser capaz de armazenar internamente o número de clientes que estão utilizando os seus serviços (esses clientes podem ser aplicações ou DLLs – chamados genericamente de processos na literatura) e implementar ao menos os seguintes métodos:

- AddRef, que incrementa em uma unidade o valor armazenado internamente (que indica o número de clientes do objeto) sempre que um novo cliente solicitar os seus serviços.
- Release, que decrementa em uma unidade o valor armazenado internamente (que indica o número de clientes do objeto) sempre que um novo cliente não precisar mais dos seus serviços (sempre que a aplicação-cliente for encerrada, por exemplo).
- QueryInterface, que informa as interfaces implementadas pelo objeto.

Esses métodos são declarados em uma interface denominada IUnknown, da qual são derivadas as demais interfaces. Pode-se, portanto, dizer que IUnknown está para uma interface assim como TObject está para uma classe.

Para facilitar a vida do programador, o Delphi declara a classe TInterfacedObject que implementa a interface IUnknown.

Essa classe é declarada da seguinte maneira na unit System.pas:

```
TInterfacedObject = class(TObject, IUnknown)
  protected
    FRefCount: Integer;
    function QueryInterface(const IID: TGUID; out Obj): Integer; stdcall;
    function _AddRef: Integer; stdcall;
    function _Release: Integer; stdcall;
  public
    property RefCount: Integer read FRefCount;
  end;
```

Um objeto pode referenciar mais de uma interface simultaneamente, o que permite simular o conceito de herança múltipla.

A seguir, apresentamos a implementação desses métodos na unit System.pas.

```
procedure TInterfacedObject.BeforeDestruction;
begin
```

```
  if RefCount <> 0 then Error(reInvalidPtr);
end;

function TInterfacedObject.QueryInterface(const IID: TGUID; out Obj): HResult;
const
  E_NOINTERFACE = $80004002;
begin
  if GetInterface(IID, Obj) then Result := 0 else Result := E_NOINTERFACE;
end;

function TInterfacedObject._AddRef: Integer;
begin
  Result := InterlockedIncrement(FRefCount);
end;

function TInterfacedObject._Release: Integer;
begin
  Result := InterlockedDecrement(FRefCount);
  if Result = 0 then
    Destroy;
end;
```

Não se preocupe se você não compreende a implementação desses métodos, pois você não precisará manipulá-los diretamente. É importante, no entanto, saber que os mesmos foram realmente implementados na classe TInterfacedObject.

Observe a definição da classe TComObject (na unit comobj) que implementa duas interfaces:

```
TComObject = class(TObject, IUnknown, ISupportErrorInfo)
  private
    FController: Pointer;
    FFactory: TComObjectFactory;
    FNonCountedObject: Boolean;
    FRefCount: Integer;
    FServerExceptionHandler: IServerExceptionHandler;
    function GetController: IUnknown;
  protected
    { IUnknown }
    function IUnknown.QueryInterface = ObjQueryInterface;
    function IUnknown._AddRef = ObjAddRef;
    function IUnknown._Release = ObjRelease;
    { IUnknown methods for other interfaces }
    function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
    function _AddRef: Integer; stdcall;
    function _Release: Integer; stdcall;
    { ISupportErrorInfo }
    function InterfaceSupportsErrorInfo(const iid: TIID): HResult; stdcall;
  public
    constructor Create;
    constructor CreateAggregated(const Controller: IUnknown);
    constructor CreateFromFactory(Factory: TComObjectFactory;
      const Controller: IUnknown);
    destructor Destroy; override;
    procedure Initialize; virtual;
    function ObjAddRef: Integer; virtual; stdcall;
    function ObjQueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
    function ObjRelease: Integer; virtual; stdcall;
    function SafeCallException(ExceptObject: TObject;
      ExceptAddr: Pointer): HResult; override;
    property Controller: IUnknown read GetController;
    property Factory: TComObjectFactory read FFactory;
    property RefCount: Integer read FRefCount;
```

```
    property ServerExceptionHandler: IServerExceptionHandler
      read FServerExceptionHandler write FServerExceptionHandler;
  end;
```

Criando um Objeto COM em Delphi

Para criar um objeto COM em Delphi, você deve executar os seguintes procedimentos:

1. Fechar todos os projetos abertos no Delphi 7.
2. Selecionar o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items.
3. Selecionar o item ActiveX Library na página ActiveX dessa caixa de diálogo.
4. Selecionar novamente o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items.
5. Selecionar o item COM Object na página ActiveX dessa caixa de diálogo.
6. Selecionar o botão Ok, para fechar essa caixa de diálogo e exibir a caixa de diálogo COM Object Wizard, mostrada na figura a seguir.

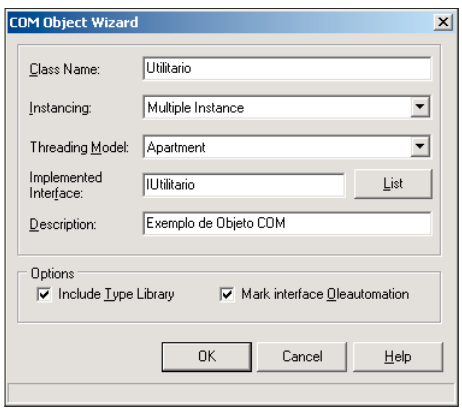


Figura 34.1: A caixa de diálogo COM Object Wizard.

7. Preencher essa caixa de diálogo como indicado na figura anterior.
8. Selecionar o botão Ok para fechar essa caixa de diálogo.

Como foi selecionada a opção Include Type Library, será exibido o Type Library Editor, mostrado na Figura 34.12.

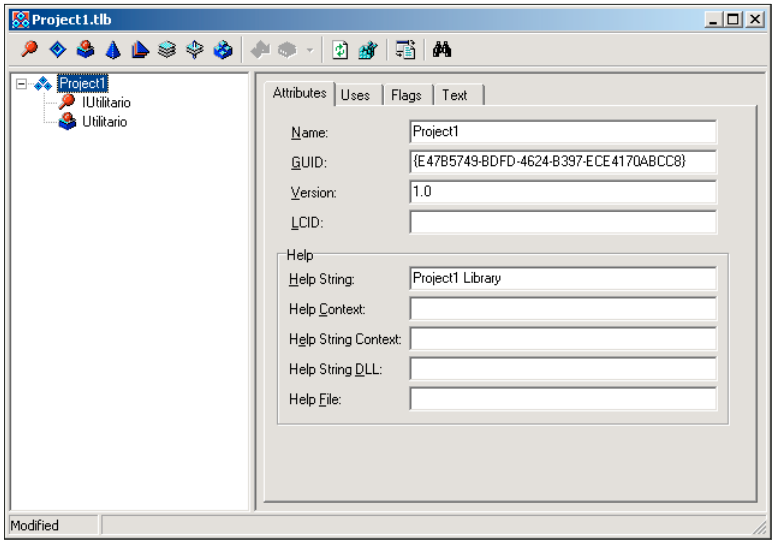


Figura 34.2: O Type Library Editor.

Adicionando e Implementando um Método à Interface IUtilitario

Para adicionar um método à interface IUtilitario, criada no tópico anterior para o nosso

objeto COM, você deve executar os seguintes procedimentos:

1. Clicar com o botão direito do mouse sobre o nome da Interface e, no menu pop-up que será exibido, selecionar o item Method, do submenu New, como mostrado na figura a seguir, ou selecionar o botão correspondente na barra de botões do Type Library Editor.

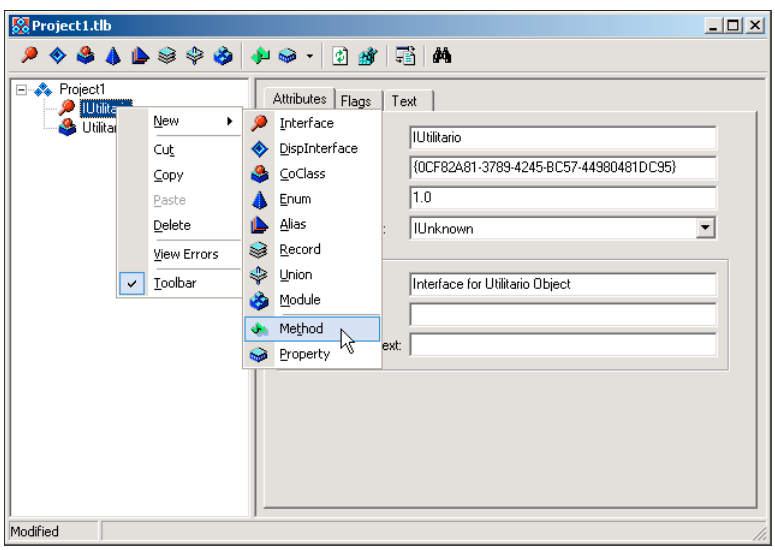


Figura 34.3: Adicionando um método a uma interface.

2. Definir um nome para o método (nesse caso denominamos esse método de GetData). Seu cabeçalho será gerado automaticamente, como pode ser visto selecionando-se a guia Text no painel direito do Type Library Editor (veja a figura a seguir).

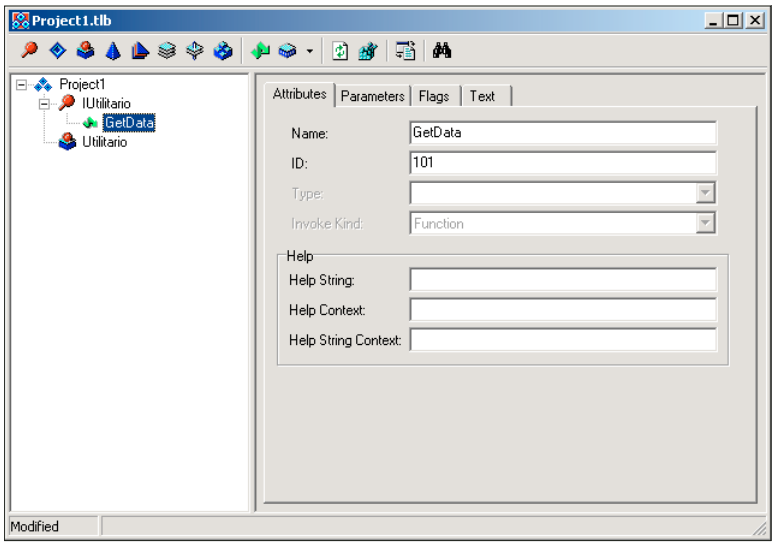


Figura 34.4: Visualizando o cabeçalho de um método no Type Library Editor.

3. Altere o tipo de retorno do método como indicado na Figura 34.5 e selecione o botão Refresh. Repare que o cabeçalho do método passa a referenciar este tipo de parâmetro, como mostrado na Figura 34.6.

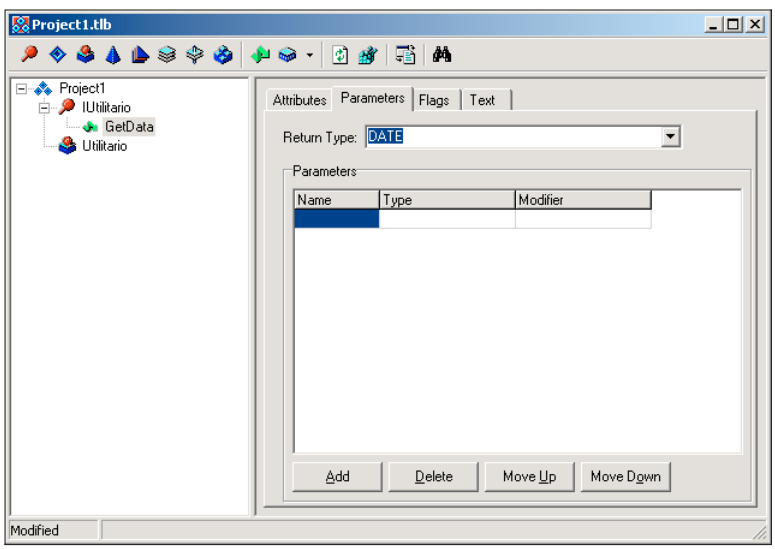


Figura 34.5: Alterando o tipo de retorno do método.

4. Selecione o botão Refresh Implementation na caixa de diálogo Type Library Editor, para gerar a implementação do método GetDate.

O Delphi gera automaticamente uma unit na qual devem ser implementados os métodos da interface e um arquivo com o mesmo nome do projeto mas acrescido com o sufixo _TLB de extensão .pas no qual é implementado o objeto COM.

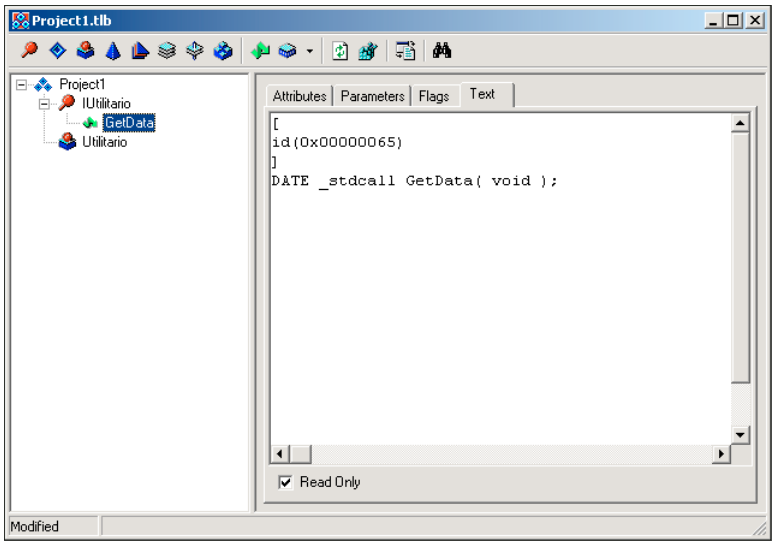


Figura 34.6: Visualizando a alteração do cabeçalho do método no Type Library Editor.

Como nesse caso foi criado um projeto novo, chamado Project1, foram então criados os arquivos Project1_TLB.pas e Unit1.pas, reproduzidos a seguir.

```
Arquivo Unit1.pas:

unit Unit1;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, ActiveX, Classes, ComObj, Project1_TLB, StdVcl;

type
  TUtilitario = class(TTypedComObject, IUtilitario)
  protected
    function GetData: TDateTime; stdcall;
    {Declare IUtilitario methods here}
  end;

implementation

uses ComServ;

function TUtilitario.GetData: TDateTime;
begin

end;

initialization
```

```
  TTypedComObjectFactory.Create(ComServer, TUtilitario, Class_Utilitario,
    ciMultiInstance, tmApartment);
end.
```

Arquivo Project1_TLB.pas:

```
unit Project1_TLB;

// ************************************************************************ //
// WARNING
// -------
// The types declared in this file were generated from data read from a
// Type Library. If this type library is explicitly or indirectly (via
// another type library referring to this type library) re-imported, or the
// 'Refresh' command of the Type Library Editor activated while editing the
// Type Library, the contents of this file will be regenerated and all
// manual modifications will be lost.
// ************************************************************************ //

// PASTLWTR : 1.2
// File generated on 21/4/2003 23:45:16 from Type Library described below.

// ************************************************************************  //
// Type Lib: C:\Arquivos de programas\Borland\Delphi7\Projects\Project1.tlb (1)
// LIBID: {E47B5749-BDFD-4624-B397-ECE4170ABCC8}
// LCID: 0
// Helpfile:
// HelpString: Project1 Library
// DepndLst:
//   (1) v2.0 stdole, (C:\WINDOWS\System32\stdole2.tlb)
// ************************************************************************ //
{$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
{$WARN SYMBOL_PLATFORM OFF}
{$WRITEABLECONST ON}
{$VARPROPSETTER ON}
interface

uses Windows, ActiveX, Classes, Graphics, StdVCL, Variants;


// *********************************************************************//
// GUIDS declared in the TypeLibrary. Following prefixes are used:
//   Type Libraries     : LIBID_xxxx
//   CoClasses          : CLASS_xxxx
//   DISPInterfaces     : DIID_xxxx
//   Non-DISP interfaces: IID_xxxx
// *********************************************************************//
const
  // TypeLibrary Major and minor versions
  Project1MajorVersion = 1;
  Project1MinorVersion = 0;

  LIBID_Project1: TGUID = '{E47B5749-BDFD-4624-B397-ECE4170ABCC8}';

  IID_IUtilitario: TGUID = '{0CF82A81-3789-4245-BC57-44980481DC95}';
  CLASS_Utilitario: TGUID = '{BA9313AC-BA71-4EBC-AB71-D2604975AD71}';
type

// *********************************************************************//
// Forward declaration of types defined in TypeLibrary
// *********************************************************************//
  IUtilitario = interface;
```

```
// ************************************************************************//
// Declaration of CoClasses defined in Type Library
// (NOTE: Here we map each CoClass to its Default Interface)
// ************************************************************************//
  Utilitario = IUtilitario;


// ************************************************************************//
// Interface: IUtilitario
// Flags:     (256) OleAutomation
// GUID:      {0CF82A81-3789-4245-BC57-44980481DC95}
// ************************************************************************//
  IUtilitario = interface(IUnknown)
    ['{0CF82A81-3789-4245-BC57-44980481DC95}']
    function GetData: TDateTime; stdcall;
  end;

// ************************************************************************//
// The Class CoUtilitario provides a Create and CreateRemote method to
// create instances of the default interface IUtilitario exposed by
// the CoClass Utilitario. The functions are intended to be used by
// clients wishing to automate the CoClass objects exposed by the
// server of this typelibrary.
// ************************************************************************//
  CoUtilitario = class
    class function Create: IUtilitario;
    class function CreateRemote(const MachineName: string): IUtilitario;
  end;

implementation

uses ComObj;

class function CoUtilitario.Create: IUtilitario;
begin
  Result := CreateComObject(CLASS_Utilitario) as IUtilitario;
end;

class function CoUtilitario.CreateRemote(const MachineName: string): IUtilitario;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_Utilitario) as IUtilitario;
end;

end.
```

Repare que a interface IUtilitario tem um método chamado GetData, que é implementado pela sua CoClass TUtilitario.

A implementação da função GetData deve ser feita da seguinte maneira no arquivo Unit1.pas:

```
function TUtilitario.GetData: TDateTime;
begin
     result := Date;
end;
```

Você deve, evidentemente, incluir a unit SysUtils na cláusula Uses da unit que implementa o método. Se isso não for feito, serão gerados erros de compilação.

A definição de propriedades é feita de maneira semelhante, bastando selecionar o botão correspondente e definir corretamente seus atributos.

Compilando e Registrando o Objeto COM no seu Sistema

Para compilar e registrar seus objetos, você deve executar os seguintes procedimentos:

1. Selecione File/Save All no menu do Delphi 7. Salve a Unit com o nome unitUtilitário e o projeto, com o nome ProjectUtilitario.
2. Compile o projeto.
3. Registre o componente, selecionando o item Register ActiveX Server do menu Run. Caso o registro seja feito com sucesso, será exibida uma mensagem indicando que o registro foi feito, como mostrado na figura a seguir.

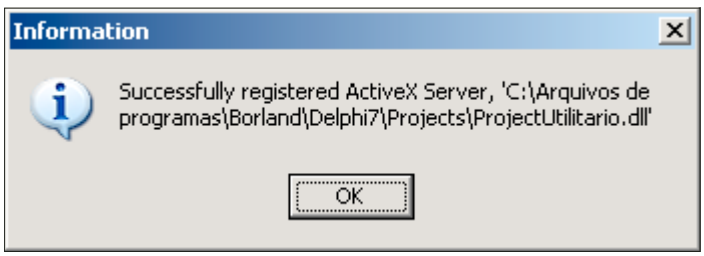


Figura 34.7: A caixa de diálogo Run Parameters.

Criando uma Aplicação que Utilize o Objeto COM

Para criar uma aplicação que use esse objeto, você deve executar os seguintes procedimentos:

1. Inicie uma nova aplicação, selecionando o item New Application do menu File.
2. Altere o valor da propriedade Name do formulário para FormData.
3. Altere o valor da propriedade Caption do formulário para 'Exemplo de Utilização de Objetos COM'.
4. Inclua um botão de comando centralizado no formulário.
5. Altere o valor da propriedade Name do botão de comando para BotaoData.
6. Altere o valor da propriedade Caption do botão de comando para 'Exibir Data'.
7. Inclua as units ComObj e PUtilitario_TLB na cláusula Uses da unit associada ao formulário.
8. Salve a unit associada ao formulário com o nome UnitUsaCOM.pas.
9. Salve o projeto com o nome ProjectUsaCOM.dpr
10. Defina da seguinte maneira o procedimento associado ao evento OnClick do botão de comando:

```
procedure TFormData.BotaoDataClick(Sender: TObject);
var
     Utilitario:IUtilitario;
begin
     Utilitario := CoUtilitario.Create;
     ShowMessage(DateToStr(Utilitario.GetData));
end;
```

Dessa maneira, quando a aplicação for executada, o objeto COM será carregado na memória e, quando o usuário selecionar o botão, a data corrente do sistema será exibida.

A Tecnologia OLE

A tecnologia OLE é, na realidade, uma extensão da tecnologia COM.

No próximo capítulo, será criada uma aplicação multicamada bastante simples, que acessa uma base de dados através de um componente TClientDataset e que utiliza a tecnologia Datasnap (antiga MIDAS), baseada na tecnologia OLE.

Capítulo 35

Aplicações Multicamadas

Neste capítulo, apresentaremos uma introdução à tecnologia Datasnap – antiga MIDAS (Multi-tier Distributed Applications Services Suite), que permite o desenvolvimento de aplicações denominadas multicamadas.

KNOW-HOW em: Aplicações Multicamadas

PRÉ-REQUISITOS

- Fundamentos da linguagem Object Pascal.
- Fundamentos da programação orientada a objetos em Delphi.
- Utilização do ambiente de desenvolvimento integrado do Delphi.
- Utilização dos componentes de acesso e visualização de bancos de dados.

METODOLOGIA

- Apresentação dos conceitos relacionados à tecnologia MIDAS, e sua implementação em Delphi.

TÉCNICA

- Apresentação dos procedimentos necessários à criação de aplicações multicamadas em Delphi.

Apresentando a Tecnologia

Uma aplicação multicamadas deve ser composta de pelo menos três camadas:

- A camada da aplicação-cliente, responsável pela interação do usuário com o sistema (também denominada camada de interface).
- A camada da aplicação servidora, também denominada aplicação intermediária, responsável por receber e enviar informações de/para a aplicação-cliente e de/para o servidor de banco de dados.
- Uma camada de armazenamento de informações, correspondendo ao servidor de bancos de dados.

A Camada de Armazenamento de Informações

A camada de armazenamento de informações é representada pelo banco de dados, o qual pode ser local ou remoto.

Neste capítulo, a fim de simplificar o entendimento por parte do leitor, e a reprodução dos exemplos apresentados, utilizaremos um banco de dados local, mas isso não compromete de forma alguma a apresentação dos conceitos fundamentais relacionados a este tópico.

Podemos por exemplo utilizar um banco de dados local, como a tabela Country.db, que acompanha o Delphi, configurada com o alias DBDEMOS.

A Camada Intermediária – a Camada Servidora

Numa aplicação multicamadas, uma aplicação-cliente jamais acessa diretamente a camada de informações – esse acesso é feito por uma aplicação servidora – à qual se podem conectar diversas aplicações clientes.

Uma aplicação servidora deve ter um objeto da classe TRemoteDataModule, no qual serão incluídos os componentes de acesso a bancos de dados, a menos que se estejam usando clientes CORBA (quando se deve usar um objeto TCORBADataModule) ou servidores definidos como DLLs instaladas com o Microsoft Transaction Server (quando se deve usar um objeto TMTSDataModule).

DCOM é um subconjunto de COM, e que se aplica à distribuição e acesso a objetos COM através de uma rede local ou da Internet.

Para criar uma aplicação servidora, você deve executar os seguintes procedimentos:

1. Iniciar um novo projeto de aplicação, selecionando o item New Application do menu File.
2. Selecionar o item New/Other do menu File para exibir a caixa de diálogo New Items.
3. Selecionar o item Remote DataModule da página MultiTier dessa caixa de diálogo.
4. Selecionar o botão Ok dessa caixa de diálogo. Será exibida a caixa de diálogo Remote Data Module Wizard, mostrada na figura a seguir, na qual deverá ser fornecido o nome do objeto a ser criado.

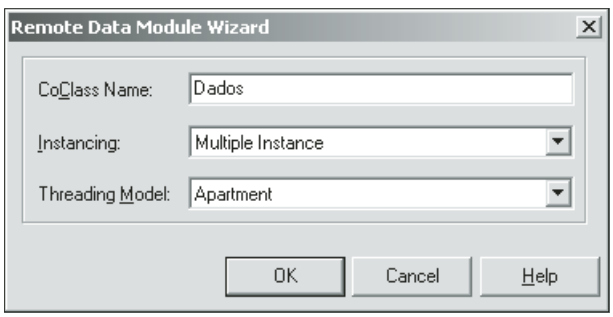


Figura 35.1: A caixa de diálogo Remote Data Module Wizard.

Neste exemplo, o objeto Remote Data Module foi denominado Dados. Repare que foram criadas duas units, denominadas Project1_TLB.pas (associada aos objetos COM a serem criados no servidor) e Unit2.pas (associada ao objeto Dados recém-criado).

5. Altere o valor da propriedade Name do formulário de Form1para FormServer.
6. Altere o valor da propriedade Caption do formulário para 'Exemplo de Aplicação Servidora'.
7. Salve a unit associada ao formulário com o nome UnitServerPrincipal.pas.
8. Coloque um componente Table (página BDE da paleta de componentes) no objeto Dados e atribua os seguintes valores para as suas principais propriedades:

   Name: TblCountry
   DatabaseName: DBDEMOS
   TableName: Country.db
   Active: True

9. Coloque um componente DatasetProvider (o terceiro componente da página Data Access da paleta de componentes) no objeto Dados e atribua os seguintes valores para as suas principais propriedades:

   Name: ProviderCountry
   Dataset: TblCountry

Seu objeto RemoteDataModule deverá ficar com o aspecto mostrado na figura a seguir.

Figura 35.2: Aspecto do objeto RemoteDataModule – Objeto Dados, após a inclusão dos objetos Table e DatasetProvider.

10. Salve a unit associada ao objeto RemoteDataModule com o nome UnitServerDados.
11. Salve o projeto com o nome ProjectServerDados. Repare que o nome da unit Project1_TLB é automaticamente modificado para ProjectServerDados_TLB, e corresponde a uma biblioteca de tipos e objetos COM.
12. Exiba a caixa de diálogo Run Parameters, selecionando o item Parameters do menu Run, e digite a expressão "/regserver" na caixa de texto Parameters, como mostrado na figura a seguir.

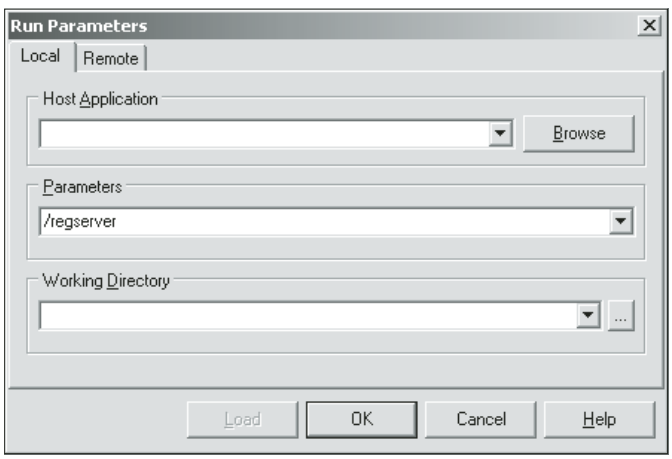


Figura 35.3: A caixa de diálogo Run Parameters.

13. Selecione o botão Ok para fechar a caixa de diálogo Run Parameters.
14. Execute a aplicação servidora, selecionando o item Run do menu Run. A aplicação será executada e encerrada imediatamente, tendo como único objetivo registrá-la no sistema.

15. Exiba novamente a caixa de diálogo Run Parameters, selecionando o item Parameters do menu Run, e deixe em branco a caixa de texto Parameters dessa caixa de diálogo.
16. Selecione o botão Ok para fechar a caixa de diálogo Run Parameters.
17. Selecione o item Build ProjectServerDados do menu Project para recompilar a aplicação servidora.

Criando a Camada de Interface com o Usuário (a aplicação-cliente)

Para criar a aplicação-cliente, você deve executar os seguintes procedimentos:

1. Iniciar um novo projeto de aplicação, selecionando o item New/Application do menu File.
2. Selecionar o item New/Other do menu File para exibir a caixa de diálogo New Items.
3. Selecionar o item DataModule da página New dessa caixa de diálogo.
4. Selecionar o botão Ok para fechar essa caixa de diálogo e criar o objeto DataModule.
5. Altere o valor da propriedade Name desse DataModule para Dados.
6. Salve a unit associada a esse DataModule com o nome UnitClienteDados.pas.
7. Altere o valor da propriedade Name do formulário criado no passo 1 para FormCliente.
8. Altere o valor da propriedade Caption desse formulário para 'Exemplo de Aplicação-Cliente'.
9. Salve a unit associada a esse formulário com o nome UnitClientePrincipal.pas.
10. Salve esse projeto com o nome ProjectCliente.dpr.
11. Inclua um componente DCOMConnection (primeiro componente da página Datasnap da paleta de componentes) no DataModule e atribua os seguintes valores às suas principais propriedades:

    Name: DCOMConnection1 (seu valor default).
    ServerName: ProjectServer.Dados (Se você registrou corretamente sua aplicação servidora, esse valor deve aparecer entre as opções disponíveis para o valor desta propriedade.)
    Connected: True

Observe que, ao se definir o valor da propriedade ServerName do componente DCOMConnection, o valor da propriedade ServerGUID é automaticamente preenchido com o identificador do objeto COM que representa o servidor. Observe ainda que, ao se definir o valor da propriedade Connected como igual a True, a aplicação servidora é inicializada (mantenha-a aberta no Windows, embora você possa minimizá-la, enquanto configura as demais propriedades da aplicação cliente).

12. Inclua um componente ClientDataset (segundo componente da página Data Access da paleta de componentes) no DataModule e atribua os seguintes valores às suas principais propriedades:

    Name: ClientDataset1 (seu valor default).
    RemoteServer: DcomConnection1 (Esse valor deve ser exibido entre as opções disponíveis para o valor desta propriedade.)
    ProviderName: ProviderCountry (Esse valor deve ser exibido entre as opções disponíveis para o valor dessa propriedade.)
    Active: True.

13. Inclua um componente DataSource no DataModule e atribua os seguintes valores às suas principais propriedades:

    Name: ClienteSource
    DataSet: ClientDataset1(Esse valor deve ser exibido entre as opções disponíveis para o valor dessa propriedade.)

Desse ponto em diante, tudo se passa como se estivéssemos em uma aplicação Desktop. De qualquer forma vamos concluir este exemplo de forma que não permaneça qualquer dúvida relativa aos conceitos apresentados.

14. Selecione o formulário principal da aplicação-cliente.
15. Selecione o item Uses Unit do menu File, para incluir a unit UnitClienteDados na cláusula Uses da unit associada a esse formulário.
16. Inclua um componente DBGrid no formulário e defina os seguintes valores para as suas principais propriedades:

    Name: DBGridCountry
    DataSource: Dados.ClienteSource

Os dados provenientes da aplicação servidora serão automaticamente visualizados no DBGrid, como mostra a figura a seguir.

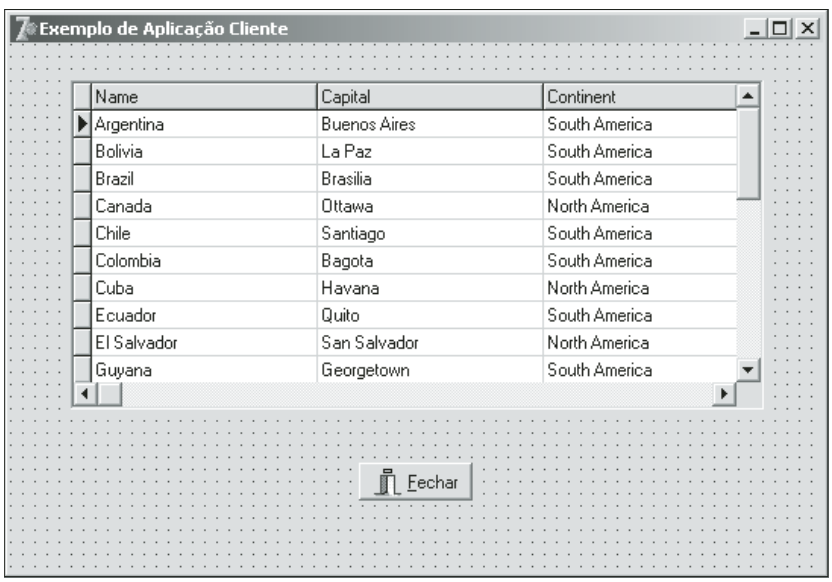


Figura 35.4: Visualizando os dados em um DBGrid.

Pronto! Sua aplicação-cliente (ainda que simplória) já pode ser executada.

O importante neste tópico é que você compreenda os conceitos de aplicações multicamadas.

Você pode definir as regras do negócio usando um objeto RemoteDataModule em uma aplicação servidora, no qual são inseridos vários componentes Table, Query, StoredProc, etc…

Evidentemente, você deve considerar o fato de que cada aplicação-cliente tem sua própria cópia dos dados (manipulados pelo componente ClientDataset), e que os conflitos resultantes do acesso de várias aplicações devem ser resolvidos nos procedimentos associados a eventos dos componentes da classe TDatasetProvider.

Embora isso não tenha sido citado explicitamente neste tópico, o desenvolvimento de aplicações servidoras também está baseado na tecnologia COM/DCOM.

Se você observar o projeto Pserver.dpr, verá que o mesmo tem uma unit denominada ProjectServerDados_TBL.pas (cujo código é reproduzido a seguir), na qual são definidos os objetos COM através dos quais se dá acesso às interfaces IProvider implementadas.

```
unit ProjectServerDados_TLB;

// ************************************************************************ //
// WARNING
// ——–
// The types declared in this file were generated from data read from a
// Type Library. If this type library is explicitly or indirectly (via
// another type library referring to this type library) re-imported, or the
// 'Refresh' command of the Type Library Editor activated while editing the
// Type Library, the contents of this file will be regenerated and all
// manual modifications will be lost.
// ************************************************************************ //

// PASTLWTR : 1.2
// File generated on 27/4/2003 18:24:51 from Type Library described below.

// ************************************************************************  //
// Type Lib: C:\Arquivos de programas\Borland\Delphi7\Projects\ProjectServerDados.tlb (1)
// LIBID: {682E06EB-EF6F-4703-B285-4ABA10FD73AF}
// LCID: 0
// Helpfile:
// HelpString: ProjectServerDados Library
// DepndLst:
//   (1) v2.0 stdole, (C:\WINDOWS\System32\stdole2.tlb)
//   (2) v1.0 Midas, (C:\WINDOWS\System32\midas.dll)
//   (3) v4.0 StdVCL, (C:\WINDOWS\system32\stdvcl40.dll)
// ************************************************************************ //
{$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
{$WARN SYMBOL_PLATFORM OFF}
{$WRITEABLECONST ON}
{$VARPROPSETTER ON}
interface

uses Windows, ActiveX, Classes, Graphics, Midas, StdVCL, Variants;

// *********************************************************************//
// GUIDS declared in the TypeLibrary. Following prefixes are used:
//   Type Libraries     : LIBID_xxxx
//   CoClasses          : CLASS_xxxx
//   DISPInterfaces     : DIID_xxxx
//   Non-DISP interfaces: IID_xxxx
// *********************************************************************//
const
  // TypeLibrary Major and minor versions
  ProjectServerDadosMajorVersion = 1;
  ProjectServerDadosMinorVersion = 0;
```

```
  LIBID_ProjectServerDados: TGUID = '{682E06EB-EF6F-4703-B285-4ABA10FD73AF}';

  IID_IDados: TGUID = '{5A36954F-937B-495D-AAAA-BFC474685F10}';
  CLASS_Dados: TGUID = '{28E32D34-63F6-42CA-AD01-23FFD01298A4}';
type

// *********************************************************************//
// Forward declaration of types defined in TypeLibrary
// *********************************************************************//
  IDados = interface;
  IDadosDisp = dispinterface;

// *********************************************************************//
// Declaration of CoClasses defined in Type Library
// (NOTE: Here we map each CoClass to its Default Interface)
// *********************************************************************//
  Dados = IDados;


// *********************************************************************//
// Interface: IDados
// Flags:     (4416) Dual OleAutomation Dispatchable
// GUID:      {5A36954F-937B-495D-AAAA-BFC474685F10}
// *********************************************************************//
  IDados = interface(IAppServer)
    ['{5A36954F-937B-495D-AAAA-BFC474685F10}']
  end;

// *********************************************************************//
// DispIntf:  IDadosDisp
// Flags:     (4416) Dual OleAutomation Dispatchable
// GUID:      {5A36954F-937B-495D-AAAA-BFC474685F10}
// *********************************************************************//
  IDadosDisp = dispinterface
    ['{5A36954F-937B-495D-AAAA-BFC474685F10}']
    function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer;
                             out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant;
dispid 20000000;
    function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut:
Integer;
                           Options: Integer; const CommandText: WideString; var Params:
OleVariant;
                           var OwnerData: OleVariant): OleVariant; dispid 20000001;
    function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
dispid 20000002;
    function AS_GetProviderNames: OleVariant; dispid 20000003;
    function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant):
OleVariant; dispid 20000004;
    function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType:
Integer;
                           var OwnerData: OleVariant): OleVariant; dispid 20000005;
    procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString;
                         var Params: OleVariant; var OwnerData: OleVariant); dispid 20000006;
  end;

// *********************************************************************//
// The Class CoDados provides a Create and CreateRemote method to
// create instances of the default interface IDados exposed by
// the CoClass Dados. The functions are intended to be used by
// clients wishing to automate the CoClass objects exposed by the
// server of this typelibrary.
// *********************************************************************//
```

```
  CoDados = class
    class function Create: IDados;
    class function CreateRemote(const MachineName: string): IDados;
  end;

implementation

uses ComObj;

class function CoDados.Create: IDados;
begin
  Result := CreateComObject(CLASS_Dados) as IDados;
end;

class function CoDados.CreateRemote(const MachineName: string): IDados;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_Dados) as IDados;
end;

end.
```

A aplicação servidora é, na realidade, um caso particular de servidor de automação OLE.

Capítulo 36

Técnicas Úteis Para a Criação da Interface com o Usuário

Neste capítulo, serão apresentadas algumas técnicas úteis, que podem simplificar o desenvolvimento da interface em aplicações desenvolvidas com o Delphi 7.

KNOW-HOW em: Parametrização de Strings de Auxílio

PRÉ-REQUISITOS

- Experiência prévia na utilização de componentes no desenvolvimento de aplicações com o Delphi 7 e na definição de strings de auxílio, utilizando-se as propriedades Hint e ShowHint dos componentes.

METODOLOGIA

- Apresentação do problema: Utilização das principais propriedades e métodos da classe TApplication na parametrização de strings de auxílio.

TÉCNICA

- Apresentação dos procedimentos necessários à parametrização de strings de auxílio em aplicações desenvolvidas com o Delphi 7.

Utilização das Strings de Auxílio (Hints)

Uma característica predominante na maioria das aplicações desenvolvidas para um ambiente com interface gráfica são as strings de auxílio, textos explicativos que são exibidos de forma a auxiliar o usuário a identificar a utilidade ou o significado de um elemento que compõe a interface da aplicação. Essa característica está presente principalmente nos botões localizados nas barras de ferramentas de diversos aplicativos.

A habilitação desse recurso é muito simples, pois basta digitar o texto desejado na propriedade Hint do componente e definir como True o valor da sua propriedade ShowHint.

Você pode, no entanto, alterar:

- A cor de fundo do texto exibido na string de auxílio.
- O tempo decorrido entre o posicionamento do mouse sobre o controle e a exibição do Hint.
- O tempo decorrido entre o início e o término da exibição de uma string de auxílio, isto é, o seu tempo de exibição.
- O tempo entre a exibição de duas strings de auxílio distintas.

Além disso, pode ainda exibir uma string de auxílio formada por várias linhas de texto.

Alterando a Cor de Fundo do Texto Exibido na String de Auxílio

Para alterar a cor de fundo do texto exibido na string de auxílio, você deve alterar o valor armazenado na propriedade HintColor do objeto Application, o que pode ser feita mediante a inclusão da seguinte linhas de código no procedimento associado ao evento OnCreate do formulário principal da aplicação:

```
Application.HintColor := clWhite;
```

O valor atribuído pode ser qualquer variável do tipo TColor ou pode ainda ser obtido utilizando-se a função RGB. Dessa maneira, a linha anterior poderia ser reescrita da seguinte maneira:

```
Application.HintColor := RGB(255,255,255);
```

A função RGB recebe como parâmetros três inteiros (cujo valor varia entre 0 e 255) e retorna a cor resultante de uma combinação das cores Vermelho (Red), Verde (Green) e Azul (Blue), nas proporções definidas pelos três valores passados como parâmetro.

Alterando o Tempo de Início e Término de Exibição da String de Auxílio

Para alterar o tempo de início e término da exibição da string de auxílio, você deve alterar o valor armazenado nas propriedades HintPause e HintHidePause do objeto Application, cujos valores default são 500 e 2500 milissegundos, respectivamente. A alteração do valor dessas propriedades pode ser feito mediante a inclusão das seguintes linha de código no procedimento associado ao evento OnCreate do formulário principal da aplicação:

```
Application.HintPause := 0;
Application.HinhidePause := 10000;
```

A partir da inclusão dessas linhas de código, as strings de auxílio serão exibidas imediatamente (pois a propriedade HintPause foi definida como sendo igual a 0) e serão exibidas por um intervalo de 10 segundos (pois o valor da propriedade HintHidePause foi definido como sendo igual a 10000 milissegundos, isto é, 10 segundos).

Alterando o Tempo de Exibição Entre Strings de Auxílio Distintas

Para alterar o tempo de exibição entre strings de auxílio distintas (de objetos distintos) você deve alterar o valor armazenado na propriedade HintShortPause do objeto Application, cujo valor default é de 50 milissegundos. A alteração do valor dessa propriedade pode ser feita mediante a inclusão da seguinte linha de código no procedimento associado ao evento OnCreate do formulário principal da aplicação:

```
Application.HintShortPause := 100;
```

Exibindo uma String de Auxílio Composta por Várias Linhas

Embora o Object Inspector só permita a digitação de strings de auxílio com uma única linha, você pode definir em código uma string de auxílio composta de várias linhas, concatenando o código ASCII da tecla Enter com várias strings em Pascal.

Por exemplo, para definir uma string de auxílio composta de duas linhas em um componente speedbutton chamado SpeedButton1, podem-se incluir as seguintes linhas de código no procedimento associado ao evento OnCreate do formulário principal da aplicação:

```
SpeedButton1.Hint := 'Linha 1'+#13#10+'Linha 2';
SpeedButton1.ShowHint := True;
```

KNOW-HOW em: Múltiplas Instâncias

PRÉ-REQUISITOS

- Experiência prévia no desenvolvimento de aplicações com Delphi 7.

METODOLOGIA

- Apresentação do problema: Definição da quantidade de instâncias de uma aplicação, que podem ser executadas simultaneamente.

TÉCNICA

- Apresentação dos procedimentos necessários à verificação de existência de múltiplas instâncias de uma aplicação.

Apresentação do Problema

Existem situações em que não se deseja a execução simultânea de múltiplas instâncias de uma aplicação.

Nesse caso, para verificar se já existe uma instância da aplicação sendo executada, deve-se utilizar a função FindWind da API do Windows.

Essa função recebe como parâmetros:

- Uma string com o nome da classe que define o formulário principal da aplicação.
- Uma string que define o caption do formulário principal da aplicação (pode ser nil).

A maneira mais fácil de impedir a execução de múltiplas instâncias de uma aplicação consiste em verificar se o valor retornado por essa função é igual a 0.

No caso de uma aplicação composta por um único formulário, da classe TFormPrincipal (que é o formulário principal da aplicação), isso pode ser feito definindo-se da seguinte maneira o arquivo de projeto da aplicação:

```
program UmaInstancia;
uses
  Windows, Dialogs,
  Forms,
  UnitPrincipal in 'UnitPrincipal.pas' {Form1};

{$R *.RES}

begin
if FindWindow('TFormPrincipal',nil) = 0
then
  begin
     Application.Initialize;
     Application.CreateForm(TFormPrincipal, FormPrincipal);
     Application.Run;
  end
else ShowMessage ('A Aplicação já está sendo executada');
end.
```

Antes de executar para testar esta aplicação no Windows, feche o seu projeto no ambiente de desenvolvimento do Delphi.

Não adianta tentar usar a variável HPrevInst, solução adotada no Delphi 1.0, pois esta não se aplica ao Windows 32 bits.

KNOW-HOW em: Reinicialização do sistema a partir de uma aplicação

PRÉ-REQUISITOS

- Experiência prévia no desenvolvimento de aplicações com Delphi 7.

METODOLOGIA

- Apresentação do problema: Reinicialização do sistema a partir de uma aplicação desenvolvida em Delphi 7.

TÉCNICA

- Apresentação dos procedimentos necessários à reinicialização do sistema a partir de uma aplicação desenvolvida em Delphi 7, baseada na VCL.

Apresentação do Problema

Muitas aplicações oferecem ao usuário a opção de reinicializar o Windows ou dar um boot no sistema após a sua instalação.

Para incluir essa característica no seu aplicativo, você deve utilizar a função ExitWindowsEx da API do Windows.

Essa função recebe como parâmetros um flag, que pode ser:

- EWX_REBOOT: Reinicializa o sistema

ou:

- EWX_SHUTDOWN: Desliga o micro.

E um segundo valor, que deve ser igual a 0.

O trecho a seguir executa a função ExitWindows e reinicializa o sistema, caso o usuário selecione o botão Ok na caixa de diálogo.

```
If MessageDlg('Deseja Fechar a Aplicação e Reinicializar o Windows
?',mtConfirmation,mbOkCancel,0)=mrOk
then ExitWindows(EW_REBOOTSYSTEM;0);
```

Dessa maneira, se o usuário selecionar o botão Ok nessa caixa de diálogo, o sistema será reinicializado.

KNOW-HOW em: Manipulação da data e hora do sistema

PRÉ-REQUISITOS

- Experiência prévia no desenvolvimento de aplicações com Delphi 7, e na codificação de aplicações em Object Pascal.

METODOLOGIA

- Apresentação do problema: Manipulação da data e hora do sistema a partir de uma aplicação desenvolvida em Delphi 7.

TÉCNICA

- Apresentação dos procedimentos necessários à manipulação da data e hora do sistema a partir de uma aplicação desenvolvida em Delphi 7.

O Tipo TDateTime

O Delphi 7 define o tipo TDateTime na unit System.pas, que é na realidade idêntico ao tipo double, como mostra a definição a seguir, extraída desta unit.

```
type TDateTime = type Double;
```

Como todo número real, uma variável do tipo TDateTime tem uma parte inteira e uma parte fracionária.

A parte inteira de uma variável do tipo TDateTime armazena o número de dias decorridos desde 30 de dezembro de 1889, até a data atual do sistema, ao passo que a parte fracionária armazena a quantidade de horas do dia.

A tabela a seguir, existente no Help On-line do Delphi, apresenta alguns valores reais e seus correspondentes no formato data-hora.

Valor Real	Data/Hora equivalente
0	30/12/1899 12:00 am
2.75	01/01/1900 6:00 pm
-1.25	29/12/1899 6:00 am
35065	01/01/1996 12:00 am

Como já foi descrito anteriormente, a parte inteira representa o número de dias decorridos desde 30/12/1899. Conseqüentemente, se você somar ou subtrair um valor inteiro a uma variável do tipo TDateTime, estará na realidade somando ou subtraindo uma determinada quantidade de dias. Além disso, pode comparar duas datas como compara dois números reais.

Para adicionar uma hora, que é 1/24 do dia, basta somar ou subtrair 1/24 à variável do tipo TDateTime.

Obtendo a Data e Hora do Sistema

O Delphi 7 tem várias funções que retornam a data e a hora do sistema, algumas das quais oriundas do velho e bom Pascal.

A função Now, por exemplo, retorna a data e a hora do sistema, ao passo que as funções Date e Time retornam apenas a data e a hora, respectivamente.

Convertendo um Valor do Tipo Data/Hora em uma String

Para converter o valor retornado por essas funções em uma string, podem-se utilizar as seguintes funções:

- TimeToStr, que recebe como parâmetro uma variável do tipo TDateTime e retorna o horário em uma string.
- DateToStr, que recebe como parâmetro uma variável do tipo TDateTime e retorna a data em uma string.
- DateTimeToStr, que recebe como parâmetro uma variável do tipo TDateTime e retorna a data e a hora em uma string.

Convertendo uma String em um Valor do Tipo Data/Hora

Para converter uma string em um valor do tipo Data/Hora, pode-se usar uma das seguintes funções fornecidas pelo Delphi 7:

- StrToDate: Converte uma string que representa uma data em uma variável do tipo TDateTime.
- StrToTime: Converte uma string que representa uma hora em uma variável do tipo TDateTime.
- StrToDateTime: Converte uma string que representa uma data/hora em uma variável do tipo TDateTime.

Obtendo o Dia da Semana Correspondente a uma Data

Para obter o dia da semana correspondente a uma data, você deve usar a função DayOfWeek, que recebe como parâmetro uma variável do tipo TDateTime e retorna um inteiro entre 1 (Domingo) e 7 (Sábado).

Funções Especiais de Conversão de Data/Hora

Suponha que você queira converter o dia 15 de fevereiro de 1999 em um valor do tipo TDateTime. Você não precisa efetuar nenhum cálculo complexo para resolver esse problema, pois o Delphi 7 fornece a função EncodeDate, que recebe como parâmetros três números inteiros positivos representando o dia (entre 1 e 31), o mês (entre 1 e 12) e o ano.

Nesse caso, por exemplo, se DataNova for uma variável do tipo TDateTime, a linha de código a ser utilizada seria:

```
NovaData := EncodeDate(15,2,1999);
```

A recíproca também é verdadeira, e você pode usar o procedimento DecodeDate para obter o dia, mês e ano correspondentes a uma data definida em uma variável do tipo TDateTime. Nesse caso, devem ser passados como parâmetros um valor do tipo TDateTime, cujo valor será convertido, e três variáveis inteiras sem sinal (tipo Word) representando respectivamente o ano, o mês e o dia a serem obtidos (estes três últimos parâmetros são passados por referência; conseqüentemente, seu valor é alterado dentro do procedimento).

A seguir, apresentamos a linha de código correspondente à utilização desse procedimento, onde DataAtual é uma variável do tipo TDateTime e Ano, Mês e Dia são variáveis do tipo Word.

```
DecodeDate(DataAtual, Ano, Mês, Dia);
```

Da mesma forma existem as funções EncodeTime e DecodeTime, que permitem a conversão em um valor do tipo TDateTime de um horário passado como hora, minuto, segundo e milissegundos (e vice-versa).

KNOW-HOW em: Personalização de Formulários com a Definição de um Pano de Fundo

PRÉ-REQUISITOS

- Experiência prévia no desenvolvimento de aplicações com Delphi 7, e na utilização dos métodos e propriedades das classes TCanvas, TBrush, TPen e TBitmap.

METODOLOGIA

- Apresentação do problema: Definição de um pano de fundo para o formulário principal de uma aplicação desenvolvida em Delphi 7.

TÉCNICA

- Apresentação dos procedimentos necessários à definição de um pano de fundo para o formulário principal de uma aplicação desenvolvida em Delphi 7.

Inserindo um Pano de Fundo em um Formulário

Para inserir um pano de fundo em um formulário, você deve definir da seguinte maneira os procedimentos associados aos seus eventos OnCreate e OnPaint:

```
procedure TFormBitmap.FormCreate(Sender: TObject);
begin
     Bitmap := TBitmap.Create;
     Bitmap.LoadFromFile('c:\windows\Egito.bmp');
end;

procedure TFormBitmap.FormPaint(Sender: TObject);
var
   X, Y, W, H : integer;
begin
   W := Bitmap.Width;
   H := Bitmap.Height;
   Y := 0;
   while Y < FormBitmap.Height do
   begin
        x := 0;
        while X < FormBitmap.Width do
        begin
              FormBitmap.Canvas.Draw(X,Y,Bitmap);
              X := X + H;
        end;
        Y := Y + H;
   end;
end;
```

Neste exemplo, a variável Bitmap deve ser declarada como um objeto do tipo TBitmap na seção var da unit do formulário.

A Figura 36.1 apresenta o resultado da execução desse código.

Figura 36.1: Preenchendo um formulário com um bitmap.

KNOW-HOW em: Desenvolvimento de Aplicações MDI

PRÉ-REQUISITOS

- Experiência prévia no desenvolvimento de aplicações com Delphi 7, e na manipulação de formulários em run-time.

METODOLOGIA

- Apresentação do problema: Desenvolvimento de aplicações MDI utilizando o Delphi 7.

TÉCNICA

- Apresentação dos procedimentos necessários ao desenvolvimento de aplicações MDI utilizando o Delphi 7 como ferramenta de desenvolvimento.

Criando Aplicações MDI

Uma aplicação MDI, ou aplicação composta por uma interface de múltiplos documentos, é uma aplicação composta por uma janela principal e várias janelas-filhas.

As aplicações MDI apresentam as seguintes características:

- As janelas-filhas não podem ser deslocadas para fora da área-cliente da janela principal (também denominada janela-pai).
- Um único menu é compartilhado entre a janela principal e as janelas-filhas.
- As janelas-filhas podem ser dispostas de uma maneira organizada a partir de uma chamada a um único método do formulário principal.
- A janela principal é responsável pela criação e destruição das janelas-filhas.

Criando a Janela Principal de uma Aplicação MDI

Para criar a janela principal de uma aplicação MDI, você deve executar os seguintes procedimentos:

1. Iniciar uma nova aplicação, selecionando o item New/Application do menu File do Delphi 7.
2. Alterar o valor da propriedade FormStyle do formulário principal para fsMDIForm.

Cada aplicação MDI só pode ter um formulário cuja propriedade FormStyle tenha o valor fsMDIForm, e esse formulário deve ser o formulário principal da aplicação.

3. Altere o valor das propriedades Name e Caption desse formulário para "FormMDIPai" e "Exemplo de Aplicação MDI".

Criando uma Janela-filha de uma Aplicação MDI

Para criar uma janela-filha para essa aplicação MDI, você deve executar os seguintes procedimentos:

1. Selecionar o item New/Form do menu File do Delphi 7, para criar um novo formulário.
2. Alterar o valor da propriedade FormStyle do formulário recém-criado para fsMDIChild.
3. Alterar os valores das propriedades Name e Caption desse formulário para "FormMDIFilho" e "Formulário filho", respectivamente.
4. Retirar esse formulário da lista Auto-Create Forms, existente na caixa de diálogo Project Options, que é exibida quando se seleciona o item Options do menu Project. No caso de aplicações MDI, normalmente cada janela-filha é criada dinamicamente quando o usuário seleciona um item de menu ou um botão de uma barra de ferramentas da janela principal.

A criação de uma janela-filha é feita mediante a inclusão da seguinte linha de código em um procedimento associado a um evento da janela-pai:

```
FormMDIFilho : TFormMDIFilho.Create(Self);
```

Nesse caso, o parâmetro Self é um ponteiro para o formulário-pai e FormMDIFilho é um objeto da classe TFormMDIFilho.

Destruindo Uma Janela-Filha de uma Aplicação MDI

A única preocupação que se deve ter ao fechar uma janela-filha consiste em liberar a memória alocada para esta janela, e isso pode ser feito atribuindo-se um valor adequado para o parâmetro Action do procedimento associado ao evento OnClose do formulário que representa a janela-filha, mediante a inclusão da seguinte linha de código nesse procedimento:

```
Action := caFree;
```

Onde Action é uma variável passada por referência no procedimento associado ao evento OnClose do formulário.

Organizando a Exibição das Janelas-filhas

Você pode organizar a disposição das janelas-filhas na área-cliente da janela-pai, e essa organização pode ser feita de diversas formas:

- Janelas-filhas dispostas lado a lado horizontalmente.
- Janelas-filhas dispostas lado a lado verticalmente.
- Janelas-filhas dispostas em cascata.

O tipo de disposição é definido atribuindo-se um valor adequado à propriedade TileMode do formulário-pai, que pode assumir os valores tbHorizontal e tbVertical.

Após definir o valor adequado para a propriedade TileMode, basta chamar o método Tile do formulário que representa a janela-pai.

Se as janelas-filhas estiverem minimizadas, você pode organizar a sua disposição executando o método ArrangeIcons da janela-pai.

Para exibir as janelas-filhas em cascata, use o método cascade da janela-pai.

Mesclando Menus

Em uma aplicação MDI, a barra de menus da janela-filha que estiver ativa substitui a barra de menus da janela-pai. Uma solução mais elegante, no entanto, consiste em mesclar as barras de menus dessas duas janelas, conforme será mostrado neste tópico.

Em geral, a barra de menus da janela principal de uma aplicação MDI apresenta os seguintes menus:

- Arquivo, com os itens Novo, Fechar e Sair.
- Janela, com as opções Lado a Lado Vertical, Lado a Lado Horizontal, Em Cascata, Organizar Ícones e Fechar Todas as Janelas.
- Ajuda, com itens que permitem acessar um arquivo de Help On-line, exibir uma caixa de diálogo de direitos autorais, etc.

As janelas-filhas, por sua vez, podem ter uma barra de menus com itens bastante específicos, dependendo da finalidade de cada janela e da aplicação propriamente dita.

Suponha que cada janela-filha tenha os seguintes menus e itens de menu, a título simplesmente ilustrativo:

- Menu Menu_A, com os itens Opção_A1, Opção_A2 e Opção_A3.
- Menu Menu_B, com os itens Opção_B1, Opção_B2 e Opção_B3.

Para que nenhum desses menus da janela-filha sobreponha um menu da janela principal, devemos configurar adequadamente o valor da propriedade GroupIndex de cada menu. A propriedade GroupIndex deve receber como valor um número inteiro, e apresenta as seguintes características:

- Se a propriedade GroupIndex de um menu da janela-filha tiver o mesmo valor de um dos menus da janela-pai, este será sobreposto.

- Os menus são exibidos, da esquerda para a direita, na ordem crescente do valor da sua propriedade GroupIndex.

Crie os menus supracitados nas janelas-pai e filha (usando o componente MainMenu), definindo os valores mostrados na tabela a seguir para a sua propriedade ItemIndex.

Menu	Propriedade GroupIndex
Arquivo (janela-pai):	0
Janela (janela-pai):	3
Ajuda (janela-pai):	4
Menu_A (janela-filha)	1
Menu_B (janela-filha)	2

Agora, quando uma janela-filha for exibida, seu menu não será mesclado com o da janela-pai.

CODIFICANDO A APLICAÇÃO

Para exibir as janelas-filhas na disposição lado a lado horizontal, basta atribuir o valor tbHorizontal à propriedade TileMode da janela principal, e executar o seu método Tile.

Para exibir as janelas-filhas na disposição lado a lado vertical, basta atribuir o valor tbVertical à propriedade TileMode da janela principal, e executar o seu método Tile.

Para exibir as janelas-filhas na disposição em cascata, basta executar o método Cascade do formulário principal.

Para fechar todas as janelas-filhas, basta executar o método Close de cada uma delas. As janelas-filhas estão referenciadas na propriedade MDIChildren, que é um a array de objetos da classe TForm, iniciando em 0. Para obter o número de janelas-filhas, basta verificar o valor da propriedade MDIChildCount da janela principal.

Apresentamos, a seguir, o código das units associadas aos formulários da aplicação.

```
unit UnitMDIPai;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus;

type
  TFormMDIPai = class(TForm)
    MainMenu1: TMainMenu;
    Arquivo1: TMenuItem;
    Novo1: TMenuItem;
    Fechar1: TMenuItem;
    Sair1: TMenuItem;
```

```
    Janelas1: TMenuItem;
    LadoaLadoHorizontal1: TMenuItem;
    LadoaLadoVertical1: TMenuItem;
    FecharTodas1: TMenuItem;
    Organizarcones1: TMenuItem;
    Ajuda1: TMenuItem;
    Sobre1: TMenuItem;
    EmCasacata1: TMenuItem;
    procedure Novo1Click(Sender: TObject);
    procedure LadoaLadoHorizontal1Click(Sender: TObject);
    procedure LadoaLadoVertical1Click(Sender: TObject);
    procedure FecharTodas1Click(Sender: TObject);
    procedure EmCasacata1Click(Sender: TObject);
    procedure Organizarcones1Click(Sender: TObject);
    procedure Sair1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormMDIPai: TFormMDIPai;
  nchilds : integer;

implementation

uses UnitMDIFilha;

{$R *.DFM}

procedure TFormMDIPai.Novo1Click(Sender: TObject);
var
   FormMDIFilho : TFormMDIFilho;
begin
     FormMDIFilho := TFormMDIFilho.Create(Self);
     nchilds := nchilds + 1;
     FormMDIFilho.Caption := FormMDIFilho.caption + ' #'+inttostr(nchilds);
end;

procedure TFormMDIPai.LadoaLadoHorizontal1Click(Sender: TObject);
begin
     Tilemode := tbHorizontal;
     Tile;
end;

procedure TFormMDIPai.LadoaLadoVertical1Click(Sender: TObject);
begin
     Tilemode := tbVertical;
     Tile;
end;

procedure TFormMDIPai.FecharTodas1Click(Sender: TObject);
var
   i : integer;
begin
   for i:=0 to MDIChildCount-1 do
        MDIChildren[i].Close;
end;

procedure TFormMDIPai.EmCasacata1Click(Sender: TObject);
begin
    Cascade;
```

```
end;

procedure TFormMDIPai.Organizarcones1Click(Sender: TObject);
begin
     ArrangeIcons;
end;

procedure TFormMDIPai.Sair1Click(Sender: TObject);
begin
     Close;
end;

procedure TFormMDIPai.FormCreate(Sender: TObject);
begin
     nchilds := 0;
end;

end.

unit UnitMDIFilha;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
  TFormMDIFilho = class(TForm)
    MainMenu1: TMainMenu;
    A1: TMenuItem;
    A11: TMenuItem;
    A21: TMenuItem;
    A31: TMenuItem;
    B1: TMenuItem;
    B11: TMenuItem;
    B21: TMenuItem;
    B31: TMenuItem;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormMDIFilho: TFormMDIFilho;

implementation

{$R *.DFM}

procedure TFormMDIFilho.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
     Action := caFree;
end;

end.
```

KNOW-HOW em: Operações de Drag-drop em Componentes

PRÉ-REQUISITOS

- Experiência prévia no desenvolvimento de aplicações com Delphi 7 e na manipulação de componentes e formulários.

METODOLOGIA

- Apresentação do problema: Desenvolvimento de aplicações que incorporam o recurso de drag-drop utilizando o Delphi 7 como ferramenta de desenvolvimento.

TÉCNICA

- Apresentação dos procedimentos necessários ao desenvolvimento de aplicações que incorporam o recurso de drag-drop.

Apresentação do Problema

O recurso de drag e drop (arrastar e soltar) pode tornar a interface de suas aplicações mais atrativa e fácil de utilizar.

Essas operações podem envolver componentes ou itens de componentes, dependendo da finalidade da aplicação, e dos objetos utilizados na criação da sua interface.

Neste tópico, serão usados componentes da página Win 3.1, inexistente na CLX. Conseqüentemente, será criada uma aplicação baseada na VCL do Delphi 7.

Descrição das Técnicas de Drag & Drop

Todo controle tem uma propriedade denominada DragMode, que controla a maneira pela qual um componente responderá às operações de drag-drop. Essa propriedade pode assumir os seguintes valores:

- dmAutomatic: Se o valor da propriedade DragMode for igual a dmAutomatic, a operação de drag-drop será iniciada assim que o usuário pressionar o botão esquerdo do mouse quando o cursor estiver posicionado sobre o componente.
- dmManual: Se o valor da propriedade DragMode for igual a dmManual (que é a situação default), a operação de drag-drop será iniciada mediante a execução de uma chamada ao método BeginDrag do componente. Essa opção deve ser empregada quando a operação de drag-drop interfere com outros eventos do mouse.

O método BeginDrag recebe como parâmetro um valor booleano, que indica se a operação de arrastar e soltar deve ou não ser iniciada imediatamente (permitindo ao componente tratar outros eventos relacionados ao mouse). Se o parâmetro for igual a False, a operação de drag-drop só será iniciada após o mouse se deslocar a uma certa distância.

Quando o usuário arrasta algum objeto sobre um controle, o evento OnDragOver desse controle é disparado, e o procedimento a ele associado é executado. Esse procedimento tem um parâmetro chamado Accept, que deverá ser definido como True caso o componente aceite receber o objeto que está sendo arrastado.

Além do parâmetro Accept, o procedimento tem os seguintes parâmetros:

- Sender, Source: Identificam respectivamente o objeto atual e aquele que iniciou a operação de drag-drop.
- X, Y: Valores inteiros que retornam a posição corrente do cursor do mouse.
- State: Variável do tipo tDragState, que pode assumir os seguintes valores:
  - dsDragEnter: o mouse acaba de atingir o controle.
  - dsDragMove: o mouse acaba de se mover sobre o controle.
  - dsDragLeave: o mouse está deixando o controle.

A operação a ser feita quando o usuário liberar o botão do mouse sobre o controle, encerrando a operação de drag-drop, deve ser definida no procedimento associado ao seu evento OnDragDrop. O procedimento associado a esse evento tem os mesmos parâmetros que o procedimento associado ao evento OnDragOver, exceto o parâmetro Accept.

Enquanto isso, no componente que originou a operação, o evento OnEndDrag é disparado assim que a operação de drag-drop termina. O procedimento associado a esse evento recebe como parâmetros:

- Sender: identifica o objeto atual.
- Target: identifica o objeto sobre o qual o objeto que estava sendo arrastado foi liberado. Se esse valor for igual a nil, indica que nenhum componente aceitou receber o objeto que estava sendo arrastado.
- X, Y: Valores inteiros que retornam a posição corrente do cursor do mouse.

A propriedade DragCursor define a imagem a ser exibida pelo cursor do mouse durante a operação de drag-drop.

Nos tópicos seguintes será apresentada uma aplicação-exemplo, na qual serão ilustradas as técnicas de drag & drop.

Exemplo de Utilização

Neste tópico, apresentaremos um exemplo de utilização das técnicas de drag & drop.

O objetivo deste exemplo será o de selecionar o nome de um arquivo-texto em uma lista de arquivos, arrastá-lo e, ao soltá-lo sobre um componente Memo, carregar neste o texto armazenado no arquivo.

Definição da Interface

Inicialmente, deverá ser definida uma nova aplicação, composta por um formulário no qual serão inseridos vários componentes, com as propriedades descritas a seguir.

- Formulário:
  Name: FormDragDrop
  Width: 545

Height: 300
Position: poScreenCenter
Caption: Exemplo de Drag & Drop

- Componente Memo:
  Name: MemoTXT
  Left: 190
  Top: 20
  Width: 235
  Height: 235
  ScrollBars: ssBoth
- Componente FileListBox:
  Name: ListaArquivos
  Left: 25
  Top: 150
  Width: 145
  Height: 97
- Componente DirectoryListBox:
  Name: ListaDiretorios
  Left: 25
  Top: 45
  Width: 145
  Height: 97
  FileList: ListaArquivos
- Componente DriveComboBox:
  Name: ListaDrives
  Left: 25
  Top: 20
  Width: 145
  Height: 19
  DirList: ListaDiretorios
- Componente Botão de Comando:
  Name: BotaoFechar
  Left: 440
  Top: 125
  Width: 75
  Height: 25
  Kind: bkClose
  Caption: &Fechar

Apague o texto exibido pela propriedade Lines do Componente MemoTXT.

Seu formulário deverá ficar com o aspecto mostrado na figura a seguir.

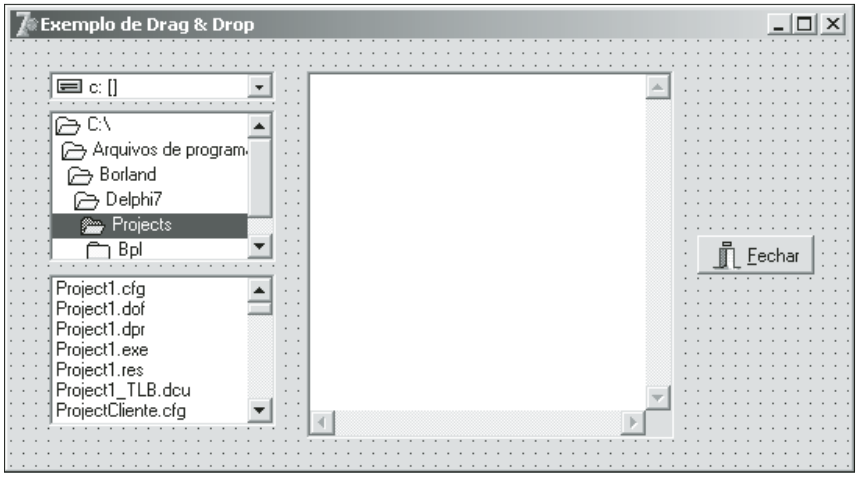


Figura 36.2: Aspecto do formulário usado no exemplo de drag & drop.

Codificação do Exemplo

A codificação deste exemplo será feita levando em consideração que o objeto, a partir do qual será iniciada a operação de drag e drop, será o componente ListaArquivos (o Source) e o componente sobre o qual este será liberado será o componente Memo (o Target).

A codificação do início do processo é feita definindo-se da seguinte maneira o procedimento associado ao evento OnMouseDown do componente ListaArquivos:

```
procedure TFormDragDrop.ListaArquivosMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     ListaArquivos.BeginDrag(False);
end;
```

O procedimento associado ao evento OnDragDrop do componente MemoTXT deve ser codificado da maneira apresentada a seguir:

```
procedure TFormDragDrop.MemoTXTDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
     if (Source is TFileListBox)
     then
         Accept := True
     else
         Accept := False;
end;
```

Esse código garante que a operação só será aceita se o objeto que iniciou o processo de drag e drop for da classe TFileListBox.

Para encerrar o processo, deve-se codificar da seguinte maneira o procedimento associado ao evento OnEndDrag do componente ListaArquivos:

```
procedure TFormDragDrop.MemoTXTDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
      if UpperCase(ExtractFileExt((Source as TFileListBox).FileName))= '.TXT'
      then
      begin
          (Sender as TMemo).Lines.LoadFromFile((Source as TFileListBox).FileName);
          ShowMessage('Arquivo Carregado com Sucesso');
      end
      else
          ShowMessage('Esta Operação só pode ser feita em arquivo-texto ASCII com extensão TXT');
end;
```

Esse procedimento verifica se o nome do arquivo cujo nome está sendo arrastado sobre o componente tem a extensão TXT e, em caso positivo, carrega o arquivo na propriedade Lines do componente.

Caso a operação tenha sido executada com sucesso, a seguinte mensagem será exibida:

```
"Arquivo Carregado com Sucesso"
```

Caso a operação não possa ser realizada, a seguinte mensagem será exibida:

```
"Esta Operação só pode ser feita em arquivo-texto ASCII com extensão TXT"
```

Apresenta-se a seguir o código da unit associada ao formulário desta aplicação-exemplo, tendo esta unit sido salva com o nome UnitDragDrop.pas.

```
unit UnitDragDrop;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, FileCtrl;
type
  TFormDragDrop = class(TForm)
    MemoTXT: TMemo;
    ListaArquivos: TFileListBox;
    ListaDiretorios: TDirectoryListBox;
    ListaDrives: TDriveComboBox;
    BotaoFechar: TBitBtn;
    procedure MemoTXTDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure MemoTXTDragDrop(Sender, Source: TObject; X, Y: Integer);
    procedure ListaArquivosMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormDragDrop: TFormDragDrop;

implementation

{$R *.DFM}

procedure TFormDragDrop.MemoTXTDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
     if (Source is TFileListBox)
```

```
      then
          Accept := True
      else
          Accept := False;
end;

procedure TFormDragDrop.MemoTXTDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
      if UpperCase(ExtractFileExt((Source as TFileListBox).FileName))= '.TXT'
      then
      begin
          (Sender as TMemo).Lines.LoadFromFile((Source as TFileListBox).FileName);
          ShowMessage('Arquivo Carregado com Sucesso');
      end
      else
          ShowMessage('Esta Operação só pode ser feita em arquivo-texto ASCII com extensão TXT');
end;

procedure TFormDragDrop.ListaArquivosMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
      ListaArquivos.BeginDrag(False);
end;

end.
```

Esse problema tem várias soluções alternativas, dependendo da estratégia empregada pelo desenvolvedor.

Capítulo 37

Internacionalização de Aplicativos Criados com o Delphi

Neste capítulo, serão apresentados os principais recursos disponíveis nesta versão do Delphi, para a internacionalização de aplicativos baseados na VCL.

KNOW-HOW em: Internacionalização de Aplicativos

PRÉ-REQUISITOS

- Experiência na utilização do ambiente de desenvolvimento integrado do Delphi 7.

METODOLOGIA

- Apresentação das principais ferramentas do Delphi 7 para a internacionalização de aplicativos.

TÉCNICA

- Apresentação dos procedimentos necessários à utilização do Translation Manager.

O Ambiente Integrado de Tradução do Delphi 7

O ambiente integrado de tradução (Integrated Translation Environment – ITE) é um conjunto de ferramentas que permite desenvolver de forma mais produtiva softwares para o mercado internacional, permitindo gerenciar o desenvolvimento de aplicações a serem comercializadas em diversas linguagens, a partir de um único projeto de aplicativo.

Este ambiente pode ser subdividido nas seguintes ferramentas:

- Translation Manager: Exibe um grid que permite a visualização e edição de recursos a serem traduzidos para diversos idiomas.
- Translation Repository: Consiste em um banco de dados que pode ser compartilhado entre vários projetos e desenvolvedores.
- Resource DLL wizard: Um assistente que gera e gerencia bibliotecas de vinculação dinâmicas para a manipulação destes recursos.

Desde a versão 5 do Delphi, podem-se compilar arquivos de recursos a partir do próprio ambiente de desenvolvimento integrado, sem que seja necessário utilizar um compilador de recursos independente (como o Resource Workshop).

O ambiente integrado de tradução cria e mantém uma DLL para cada versão de idioma da aplicação que está sendo desenvolvida. Estas DLLs são organizadas em um grupo de projetos, sendo acessadas através do gerenciador de projetos do Delphi.

Incorporando os Recusos do Ambiente Integrado de Tradução ao seu Projeto de Aplicativo

Para incorporar os recursos do ambiente integrado de tradução a um projeto de aplicativo, você deve executar os seguintes procedimentos:

1. Salvar o projeto atual.

2. Selecionar o item New/Other do menu File do Delphi 7 para exibir a caixa de diálogo New Items e, nessa caixa de diálogo, selecionar o item Resource DLL Wizard.

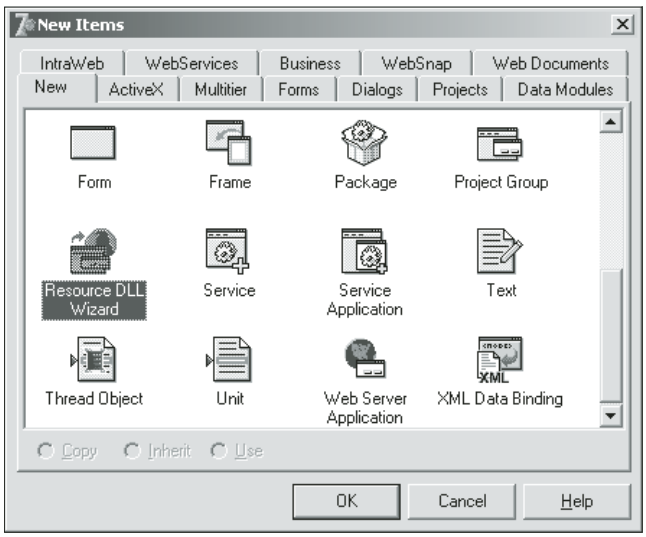


Figura 37.1: Selecionando o item Resource DLL Wizard na caixa de diálogo New Items.

3. Selecione o botão Ok para fechar esta caixa de diálogo e inicializar o assistente, conforme mostrado na figura a seguir.

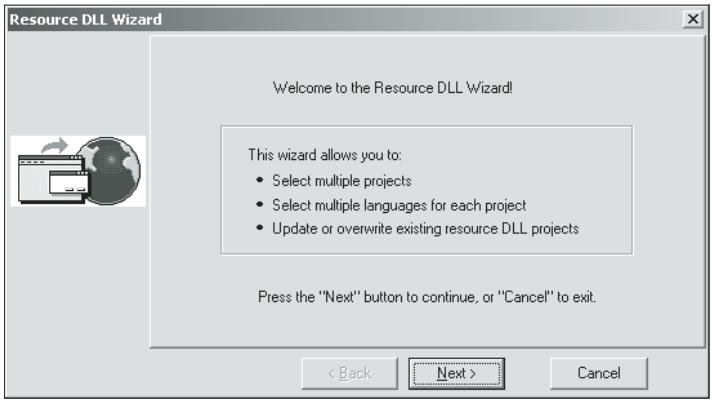


Figura 37.2: A tela inicial do Resource DLL Wizard.

Esta tela informa que você poderá:

- Selecionar múltiplos projetos.
- Selecionar múltiplas linguagens para cada projeto.
- Atualizar ou sobrecarregar as DLLs de recursos já existentes.

4. Selecione o botão Next. Será exibida uma caixa de diálogo na qual deverão ser selecionados os projetos aos quais as DLLs de recurso se aplicam, como mostrado na figura a seguir.

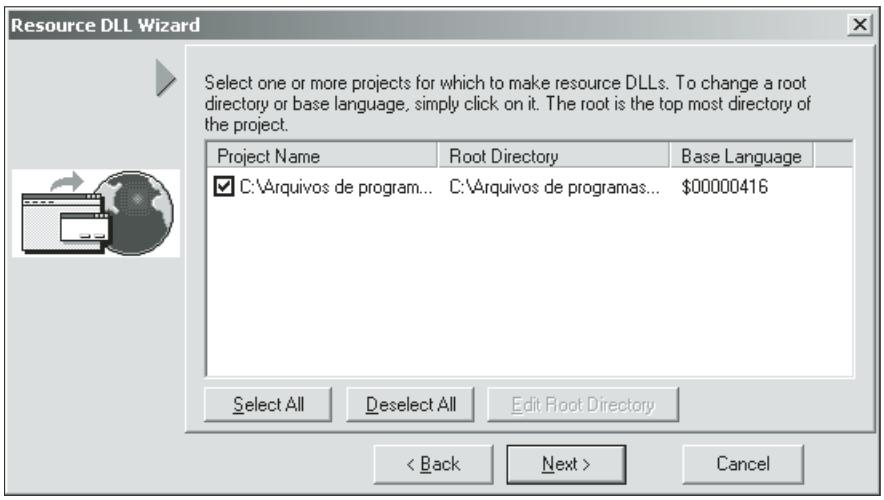


Figura 37.3: Selecionando projetos no Resource DLL Wizard.

5. Selecione o botão Next. Será exibida uma caixa de diálogo na qual deverão ser selecionadas as linguagens com as quais se deseja trabalhar, como mostrado na figura a seguir.

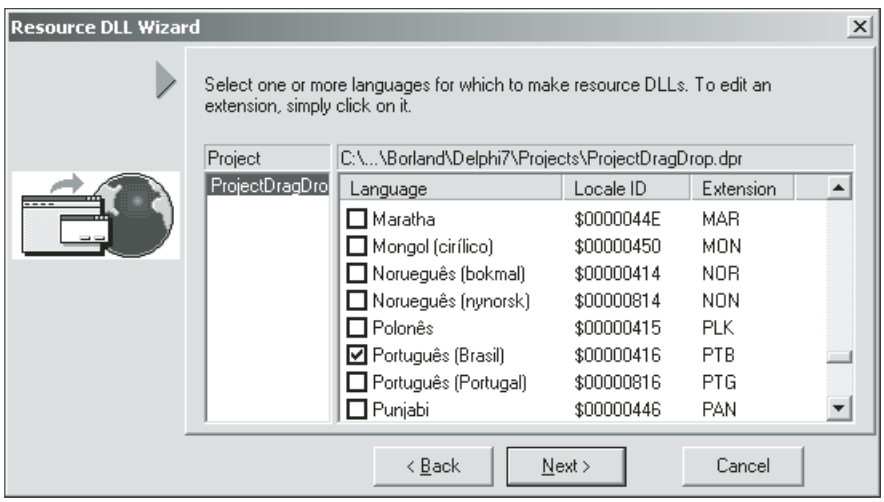


Figura 37.4: Selecionando as linguagens com as quais se deseja trabalhar.

6. Selecione o botão Next. Será exibida uma caixa de diálogo na qual serão informadas as linguagens selecionadas, e onde você poderá definir o diretório em que cada DLL será armazenada (neste exemplo foram selecionadas as opções de inglês dos Estados Unidos e Português Brasileiro) como mostrado na Figura 37.5.

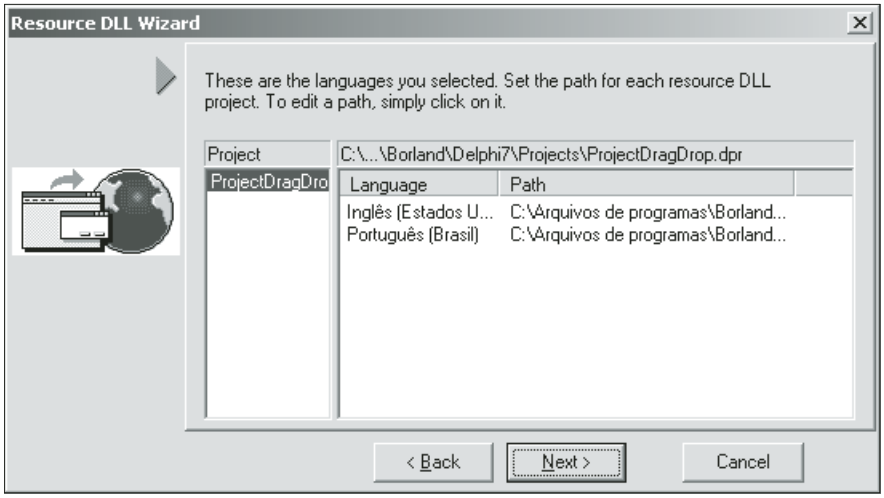


Figura 37.5: Selecionando as linguagens com as quais se deseja trabalhar.

7. Selecione o botão Next. Será exibida a caixa de diálogo mostrada na figura a seguir, na qual poderão ser informados, para cada um dos projetos, arquivos sobre os quais as DLLs de recursos também deverão atuar.

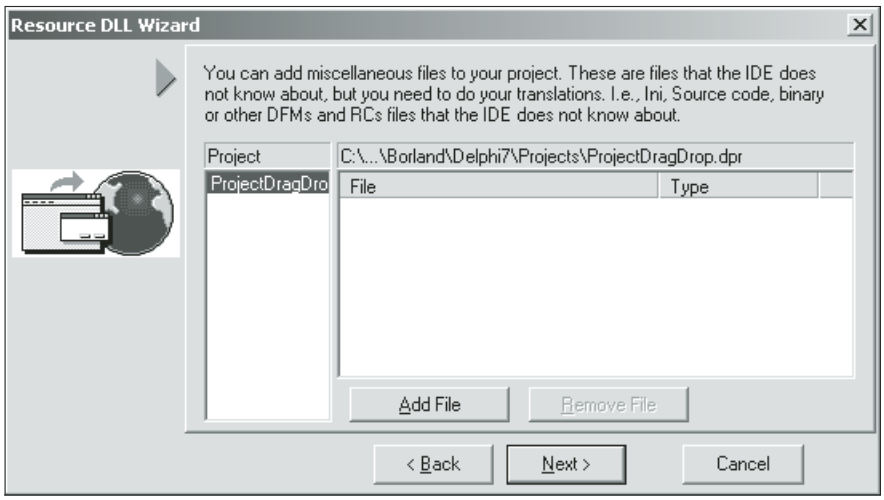


Figura 37.6: Caixa de diálogo para inclusão de arquivos adicionais.

8. Selecione os arquivos a partir da caixa de diálogo exibida quando se seleciona o botão Add File e o botão Next. Será exibida a caixa de diálogo mostrada na figura a seguir, na qual deverá ser informado o modo de atualização das DLLs de recursos – podendo optar por criar novos recursos ou atualizar os já existentes.

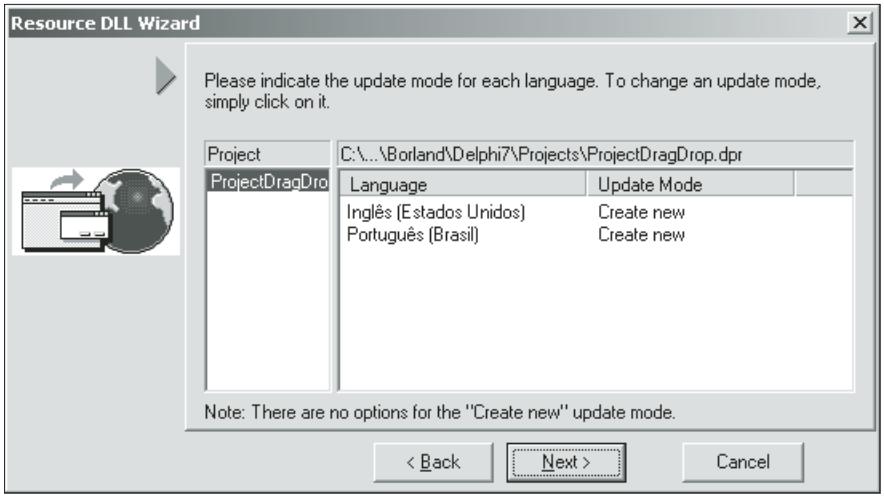


Figura 37.7: Caixa de diálogo para definição do modo de atualização das DLLs de recursos.

9. Selecione o botão Next. Será exibida a caixa de diálogo mostrada na Figura 37.8, na qual serão informadas as providências que serão realizadas pelo assistente.

10. Selecione o botão Finish. Será exibida a caixa de diálogo mostrada na Figura 38.9, na qual o usuário deve informar se o(s) projeto(s) podem ser recompilados, para que o Resource DLL Wizard seja concluído com êxito.

11. Selecione o botão Yes. Será exibida a caixa de diálogo mostrada na Figura 38.10, na qual é apresentado o resultado da compilação.

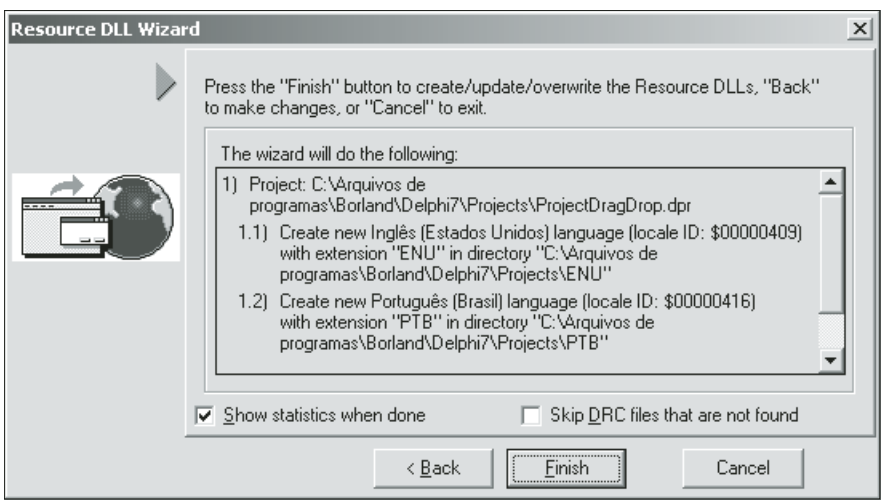


Figura 37.8: Caixa de diálogo final do Resource DLL Wizard.

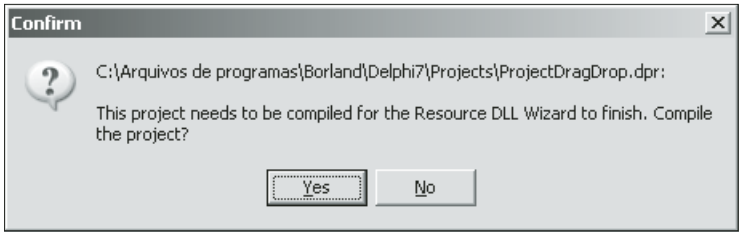


Figura 37.9: Caixa de diálogo exibida para solicitar a recompilação de projetos.

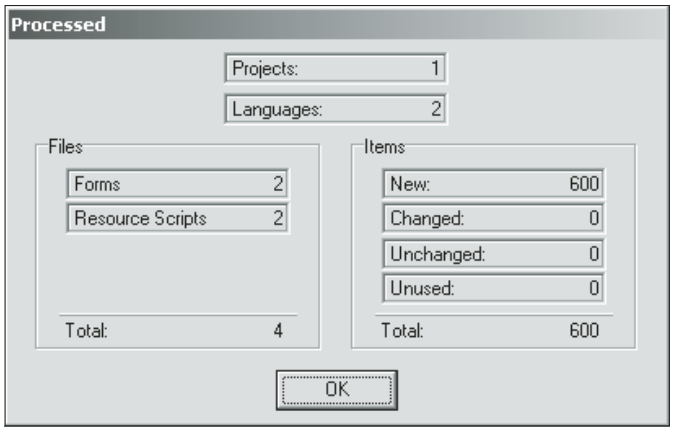


Figura 37.10: Caixa de diálogo com informações relativas ao resultado da compilação.

12. Selecione o botão Ok para fechar esta caixa de diálogo. Será exibida a caixa de diálogo Translation Manager – Gerenciador de Traduções – mostrada na Figura 37.11.

Posteriormente, para visualizar o Translation Manager, basta selecionar o item Translation manager no menu view do Delphi. Se os arquivos de recursos estiverem sendo criados pela primeira vez, será necessário salvar os arquivos do grupo de projetos que estará sendo criado automaticamente.

Traduzindo Constantes e Expressões

A tradução de constantes e expressões é feita na janela do Translate Manager.

Do lado esquerdo da guia Workspace da janela do Translate manager estão dispostos, de forma hierárquica, os elementos e arquivos de recursos correspondentes a cada um dos idiomas com os quais você resolveu trabalhar.

Os arquivos de recurso do Delphi apresentam as principais constantes e mensagens de erro do ambiente, e também podem ser traduzidas.

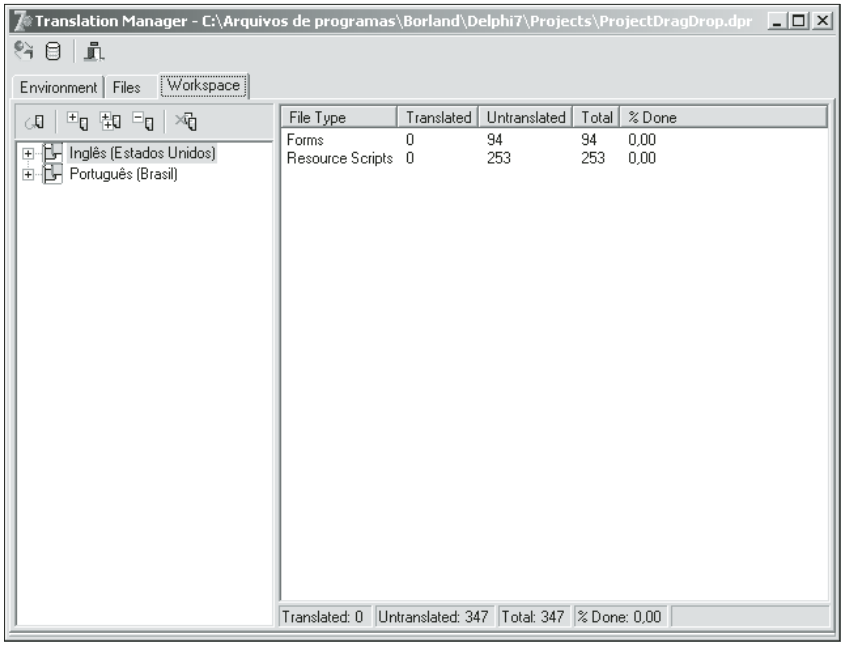


Figura 37.11: O Translation Manager.

A figura a seguir mostra a seleção do arquivo de recursos para a língua portuguesa, com algumas das strings já traduzidas.

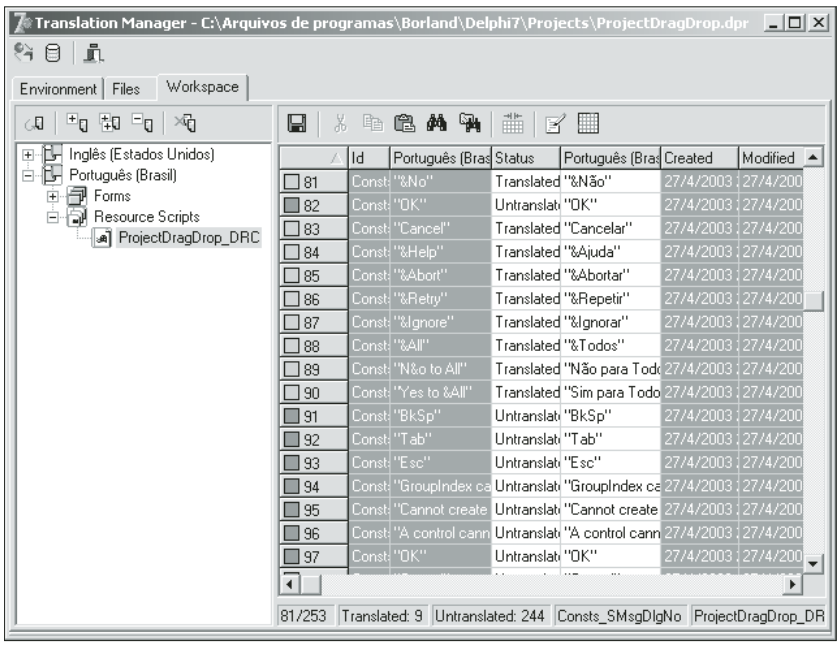


Figura 37.12: Traduzindo strings do ambiente de desenvolvimento integrado do Delphi 7.

Após fazer as traduções necessárias, você deve selecionar os botões Save (com um desenho de um disquete) e Refresh para atualizar e salvar o seu projeto.

O mesmo pode ser feito com propriedades e atributos de objetos, conforme mostrado na figura a seguir.

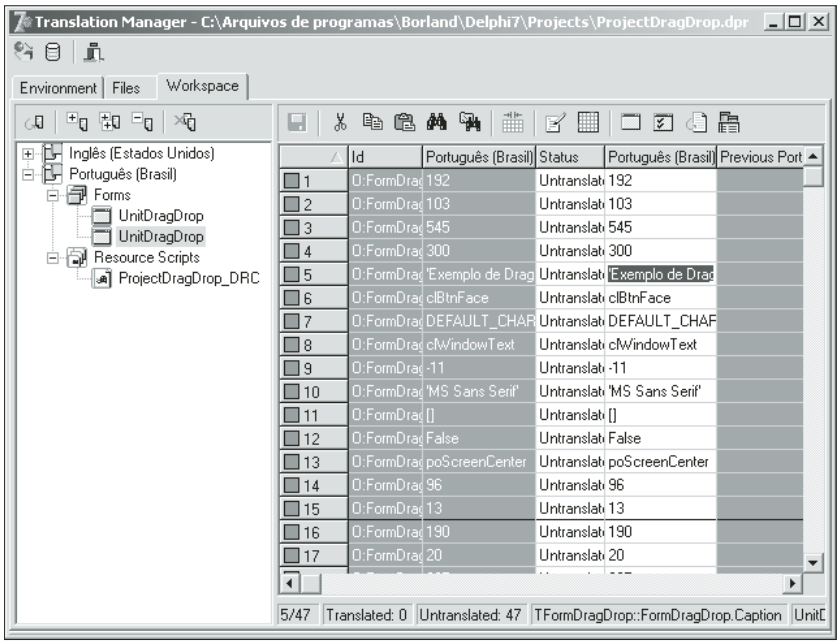


Figura 37.13: Traduzindo a propriedade Caption de um formulário.

Definindo o Idioma Corrente

Para definir o idioma corrente de um projeto de aplicação, você deverá executar um dos seguintes procedimentos:

1. No menu do Delphi 7 selecione Run/Languages/Set Active. Será exibida a caixa de diálogo mostrada na figura a seguir, na qual uma das linguagens deverá ser selecionada.

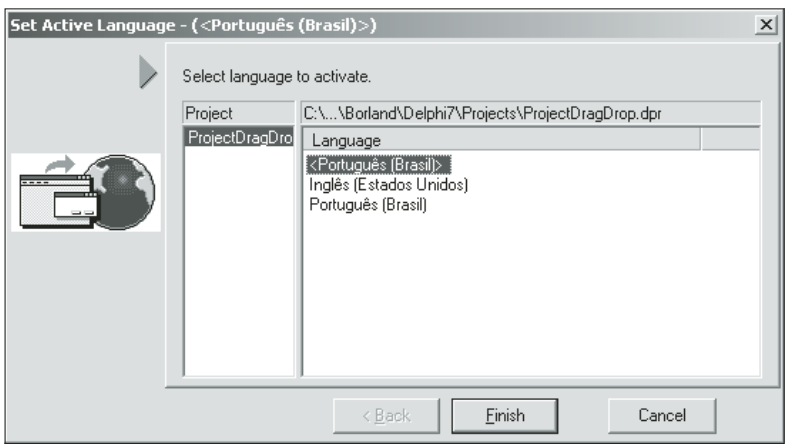


Figura 37.14: Selecionando o idioma ativo.

2. Selecione o idioma desejado e o botão Finish.

Ao selecionar um idioma ativo, o projeto será compilado com os recursos e strings definidos para o mesmo.

Utilizando o Translation Repository

O Translation Repository, conforme descrito anteriormente, é um banco de dados que pode ser compartilhado entre vários projetos e desenvolvedores.

Você pode acessar o Translation Repository selecionando o primeiro botão na barra de ferramentas superior do Translate Manager ou a partir do item Translation Repository do menu Tools.

Você pode adicionar recursos ao Translation Repository a partir da grade de strings exibida quando se seleciona um dos arquivos de recursos.

Para transferir uma string do arquivo de recursos para o Translation Repository, você deve executar os seguintes procedimentos:

1. Selecionar o arquivo de recursos na janela do Translation Manager.

2. Selecione as linhas referentes às strings a serem transferidas para o Translation Repository e pressione o botão direito do mouse. No menu pop-up que será exibido, selecione o item Repository/Add Strings to Repository.

Para transferir uma string do Translation Repository para o arquivo de recursos, você deve executar os seguintes procedimentos:

1. Selecionar o arquivo de recursos na janela do Translation Manager.

2. Selecionar o botão direito do mouse. No menu pop-up que será exibido, selecione o item Repository/Get Strings From Repository.

O Translation Repository também permite a exportação e importação de arquivos com as strings desejadas.

Você também pode selecionar diversas opções de configuração do Translation Manager na caixa de diálogo Translation Tools Options, exibida quando se seleciona o primeiro botão da barra de ferramentas superior do Translation Manager, e reproduzida na figura a seguir.

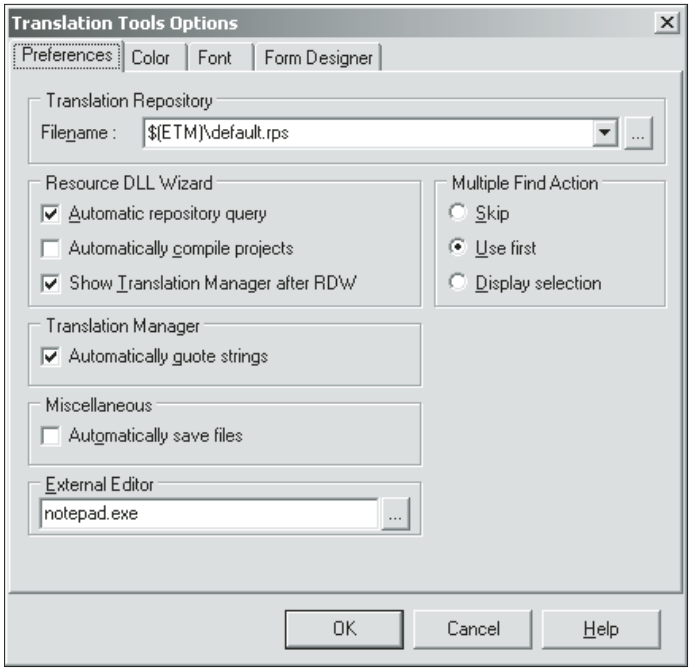


Figura 37.15 – A página Translation Tools da caixa de diálogo Environment options.

Capítulo 38

Criando Aplicações Para a Internet

A Internet, que inicialmente foi criada com fins militares e acadêmicos, teve seu principal foco totalmente alterado nos últimos anos, e passou a ter uma importância comercial indiscutível. A Internet está cada vez mais presente no nosso cotidiano, e a demanda por aplicações voltadas ao comércio eletrônico cresce a cada dia. Você provavelmente já visitou algum site através do qual podem ser realizadas compras de livros, CDs e outros produtos.

Cresce também a demanda por serviços eletrônicos das mais diversas formas, ainda existindo um mercado de serviços a serem explorados.

Neste capítulo, serão apresentados os principais recursos disponíveis para o desenvolvimento de aplicações para a Internet com o Delphi 7, utilizando-se o recurso das aplicações CGI (que poderiam inclusive ser desenvolvidas em outras linguagens – como o Pearl, C, C++ e Visual Basic), da tecnologia Websnap e da Intraweb, recentemente incorporada ao ambiente.

KNOW-HOW em: Desenvolvimento de Aplicações CGI com WebBroker

PRÉ-REQUISITOS

- Experiência em programação estruturada com versões anteriores da linguagem Pascal.
- Conhecimentos básicos da linguagem HTML.

METODOLOGIA

- Apresentação do problema: Criação de uma aplicação CGI com a tecnologia WebBroker, utilizando os componentes disponíveis no ambiente de desenvolvimento integrado do Delphi 7.

TÉCNICA

- Apresentação dos procedimentos necessários ao desenvolvimento de aplicações CGI.

Procedimentos Básicos Necessários à Criação de Aplicações CGI

Neste tópico, serão apresentados os procedimentos necessários à criação de aplicações CGI, isto é, aplicações que podem ser executadas em um servidor Web e retornar uma resposta ao browser do cliente que solicitou a execução da aplicação.

Embora o desenvolvimento de uma aplicação CGI com a tecnologia WebBroker seja mais trabalhoso do que o desenvolvimento baseado nas tecnologias Websnap e Intraweb, ele permite um maior controle do sistema por parte do desenvolvedor, e em muitas situações é, sem dúvida, a melhor solução.

Em nossos exemplos serão criadas aplicações CGI padrão, embora os procedimentos a serem utilizados para criar aplicações ISAPI e Win-CGI sejam idênticos. Foi utilizado um provedor comercial, capaz de oferecer suporte não apenas ao desenvolvimento de aplicações CGI, mas também de aplicações baseadas em Websnap e Intraweb, abordadas posteriormente.

As aplicações CGI, no entanto, podem ainda ser testadas usando o Microsoft Personal Web Server, que apresenta a vantagem de que a sua estrutura de diretórios é semelhante à existente em servidores Web baseados no sistema operacional Windows NT. Já aplicações baseadas em Websnap e Intraweb podem ser testadas a partir do próprio ambiente de desenvolvimento do Delphi 7. Conseqüentemente, uma

aplicação CGI desenvolvida localmente (na fase de testes) pode ser facilmente transferida para o seu verdadeiro servidor Web (do seu provedor).

Caso utilize o Personal Web Server, e tendo sido efetuada a sua instalação padrão, os aplicativos CGI são armazenados no diretório C:\Inetpub\wwwroot\cgi-bin, e definiu-se o DNS como localhost e o endereço IP como 127.0.0.1.

Você pode ainda usar outro servidor Web (como o Apache) e executar procedimentos semelhantes, devendo inicialmente verificar o tipo de aplicação CGI capaz de ser executada pelo seu servidor e o diretório em que essas aplicações deverão ser armazenadas (consulte o seu provedor para obter estas informações).

Uma Aplicação CGI Bastante Elementar

Neste tópico, mostraremos uma aplicação CGI extremamente simples, com o objetivo de apresentar os conceitos fundamentais cujo conhecimento é indispensável ao desenvolvimento de aplicações CGI com o Delphi 7.

Essa aplicação será capaz de gerar como resposta uma página HTML estática extremamente simples, mas que lhe ajudará a compreender os conceitos básicos. É claro que uma aplicação CGI que gere uma página HTML estática não é muito interessante (e nem precisamos de uma aplicação CGI para fazer isso), mas os conceitos fundamentais são mais fáceis de serem assimilados através de um exemplo simples, porém conceitualmente correto.

Para criar essa aplicação CGI, proceda da seguinte maneira:

1. Selecione o item New/Other do menu File do Delphi 7 para exibir a caixa de diálogo New Items.
2. Selecione o item Web Server Application nessa caixa de diálogo, como mostrado na Figura 38.1.

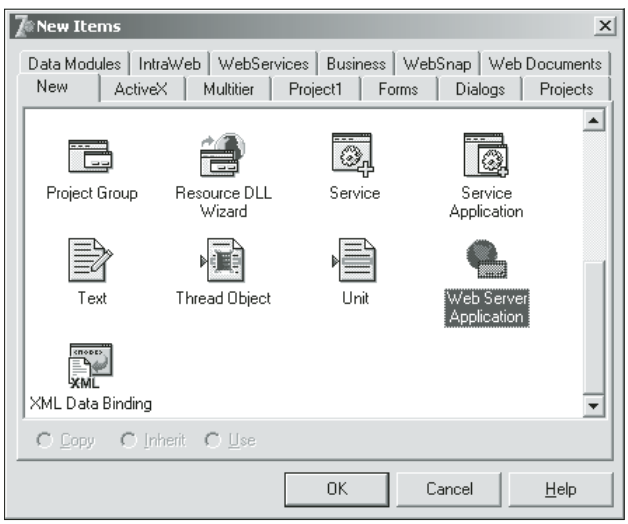


Figura 38.1: Selecionando o ícone Web Server Application na caixa de diálogo New Items.

3. Selecione o botão OK para fechar essa caixa de diálogo. Será exibida a caixa de diálogo New Web Server Application, mostrada na Figura 38.2, na qual você deve escolher o tipo de aplicação CGI capaz de ser executada pelo seu servidor Web. No caso desse exemplo, conforme já indicado anteriormente, será selecionada a opção CGI Stand-alone executable.

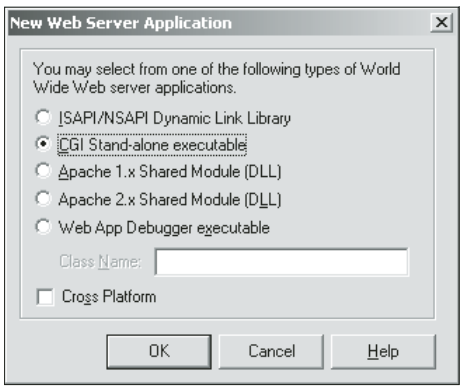


Figura 38.2: A caixa de diálogo New Web Server Application.

Caso você queira que esta aplicação CGI possa ser compilada para várias plataformas, marque o Checkbox correspondente. Neste caso, no entanto, terá dificuldades em seguir os exemplos que usam o acesso via BDE.

4. Selecione o botão OK para fechar essa caixa de diálogo. Será criada uma janela semelhante a um DataModule, mas denominada WebModule1, conforme a Figura 38.3.

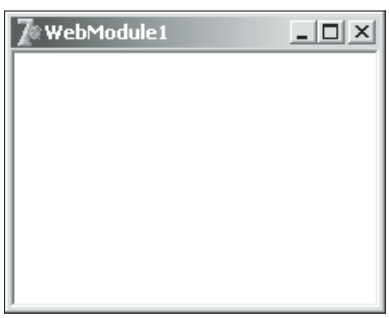


Figura 38.3: A criação do WebModule.

Um WebModule é muito semelhante a um DataModule (e, conforme será descrito posteriormente, um DataModule pode ser facilmente transformado em um WebModule mediante a inclusão de um componente WebDispatcher).

Da mesma maneira que um DataModule, um WebModule serve como um repositório para componentes não visuais, como TTable, TQuery, TADOTable, TADOQuery, TSQLTable, TSQLQuery, TPageProducer, etc.

Além disso, esse componente é responsável por definir como a aplicação deve responder a solicitações feitas por clientes através de mensagens que seguem o protocolo HTTP.

O gerenciamento dessas respostas é feito através de um "Editor de Ações", que define as respostas a serem fornecidas para diversas solicitações. No caso de esta aplicação-exemplo ser testada com o Personal web Server e salva com o nome WebCGITeste01.exe no diretório C:\inetpub\wwwroot (o projeto será salvo como WebCGITeste01.dpr), a mensagem transmitida pelo browser ao servidor decorre da digitação do seguinte endereço (URL) no browser:

```
http://localhost/WebCGITeste01.exe/parâmetros.
```

Conforme descrito anteriormente, no caso de a aplicação estar sendo disponibilizada na Web você deve substituir "localhost" pelo endereço do seu servidor (nesse caso, localhost corresponde a um servidor que está sendo executado localmente, na mesma máquina do cliente). O diretório deverá ser substituído pelo diretório configurado pelo seu servidor como repositório de aplicações CGI. No meu caso, tive que substituir "localhost" por "marceloleao.com.br/cgi-bin".

WebCGITeste01.exe é, obviamente, o nome da nossa aplicação CGI.

Os parâmetros, conforme será descrito posteriormente, informam à aplicação CGI o tipo de resposta a ser fornecida.

Para exibir o Editor de Ações (onde serão definidas as respostas da aplicação CGI), proceda da seguinte maneira:

1. Selecione o WebModule criado anteriormente e pressione o botão direito do mouse para exibir o seu menu suspenso.
2. Selecione o item Action Editor desse menu para exibir a caixa de diálogo mostrada na figura a seguir, na qual deverão ser definidas as diversas ações a serem respondidas pela aplicação CGI.

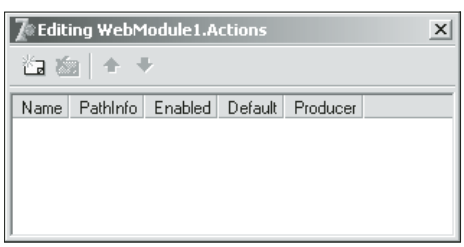


Figura 38.4: O editor de ações do WebModule1.

Inicialmente, como era de se esperar, não existe qualquer ação de resposta definida pela aplicação. Vamos então definir uma primeira ação, executando os seguintes procedimentos:

1. Selecione o botão Add New (o primeiro no canto superior esquerdo) na caixa de diálogo do Editor de Ações para criar um item chamado WebActionItem1. Esse item é, na realidade, um objeto da classe THTTPActionItem, e se você selecioná-lo poderá visualizar suas propriedades no Object Inspector.

Opcionalmente, você pode usar a tecla Ins do seu teclado ou pressionar o botão direito do mouse sobre o Editor de Ações e, no menu pop-up que será exibido, selecionar o item Add.

Na realidade, cada item da caixa de diálogo do Editor de Ações é um dos elementos da array Actions (que é uma propriedade do objeto WebModule). Por essa razão, a caixa de seleção de objetos do Object

Inspector indica o item WebModule1.Actions[0] – mostrando que WebActionItem1 é o primeiro elemento dessa array, cujo índice começa em 0.

As principais propriedades desse objeto são:

- Name: Identifica o nome pelo qual o objeto é referenciado no código.
- Default: Indica se esse item representará ou não a resposta default da aplicação. Caso nenhum dos itens satisfaça às especificações descritas na URL, a ação definida para o item default será executada (repare que no WebActionItem recém-criado sua propriedade default foi automaticamente definida como True).
- MethodType: Define os tipos de solicitações a serem atendidas e pode ter um dos valores apresentados a seguir: mtGet, mtHead, mtPost, mtPut e mtAny. O significado de cada um desses valores será descrito posteriormente e, inicialmente, utilizaremos o valor default mtAny, que aceita todos os tipos de solicitação.
- PathInfo: Esta propriedade é uma string que define o texto correspondente ao digitado na URL logo após o nome da aplicação CGI.

A URL definida a seguir, por exemplo, executará na aplicação armazenada no Personal web Server a ação definida pelo item cuja propriedade PathInfo é igual a /Exemplo.

```
http://localhost/WebCGITeste01.exe/ Exemplo
```

- Enabled: Esta propriedade define se o item está ou não habilitado. Se seu valor for igual a False, esse item será ignorado, ainda que suas propriedades MethodType e PathInfo correspondam às definidas pela URL que disparou a execução do aplicativo CGI. A única exceção ocorre quando sua propriedade default é igual a True, pois nesse caso o valor da sua propriedade Enabled é ignorado.

Para o item recém-criado, defina o valor da sua propriedade PathInfo como Exemplo. Repare que esse valor será exibido na caixa de diálogo do Editor de Ações.

Para definir uma resposta da aplicação correspondente a esse item, você deve executar os seguintes procedimentos:

1. Selecionar o item WebActionItem1 no Editor de Ações.
2. Selecionar a guia Events no Object Inspector.
3. Dar um duplo clique com o botão esquerdo do mouse sobre o espaço em branco à direita do evento OnAction. Será exibido o corpo principal do procedimento associado a esse evento, como mostrado a seguir.

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin

end;
```

Esse método apresenta os seguintes parâmetros:

- Sender: TObject. Este parâmetro identifica o objeto que disparou o evento.

- Request: TwebRequest. Este parâmetro é, na realidade, um objeto da classe TwebRequest que tem algumas propriedades que definem parâmetros da mensagem. Inicialmente, não utilizaremos esse parâmetro.
- Response: TwebResponse. Este parâmetro é um objeto da classe TwebResponse e define a resposta que será enviada ao cliente que enviou a mensagem ao servidor (através da URL). A principal propriedade desse objeto é a propriedade Content, que é uma string que define a página HTML a ser gerada como resposta à mensagem enviada pelo cliente.
- Handled: Boolean. Este parâmetro é passado por referência e define se a mensagem foi ou não completamente tratada pela aplicação.

Ao definir a propriedade PathInfo como Exemplo, indicamos que esse item responderá à mensagem representada pela seguinte URL:

```
http://localhost/ WebCGITeste01.exe/Exemplo
```

Lembre-se que, se a propriedade default do WebActionItem for mantida como True, será desnecessário incluir o PathInfo na URL.

Devemos agora definir a resposta que será enviada ao cliente por esse item, e para isso precisamos estabelecer adequadamente o valor da propriedade Content do objeto Response (que é uma string).

Nesse caso, pode-se incluir a seguinte linha de código no procedimento recém-criado:

```
Response.Content := '<HTML><BODY><H1>Exemplo de Aplicação CGI</H1></ BODY ></ HTML > ';
```

Esse procedimento passará então a ser definido como:

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
        Response.Content := '<HTML><BODY><H1>Exemplo de Aplicação CGI</H1></ BODY ></ HTML > ';
end;
```

Salve seu projeto com o nome WebCGITeste01 no diretório C:\inetpub\wwwroot (se você estiver usando o PWS) ou no diretório a partir do qual está sendo desenvolvido seu site (e a partir do qual será feito o upload dos arquivos), compile a aplicação (não a execute a partir do ambiente ou do Windows, apenas compile) e armazene-a no diretório correto. Em seguida, execute a aplicação CGI a partir de um browser, digitando a seguinte URL (no caso do PWS):

```
http://localhost/WebCGITeste01.exe/Exemplo
```

Você verá o resultado mostrado na Figura 38.5.

Nesta figura usei o endereço do provedor no qual fiz os testes:

```
http://www.marceloleao.com.br/cgi-bin/WebCGITeste01.exe/Exemplo
```

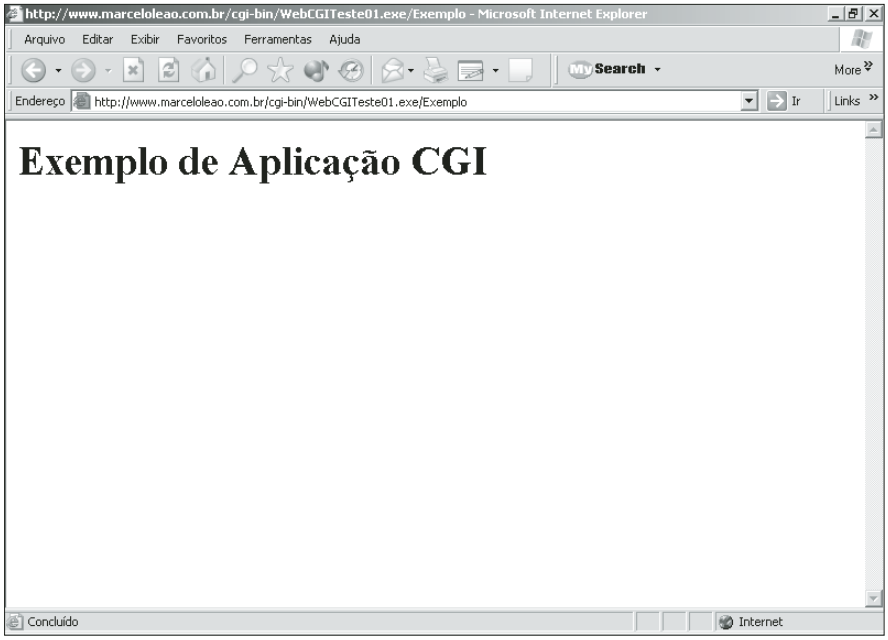


Figura 38.5: Visualizando o resultado no Inernet Explorer.

Conforme já descrito anteriormente, caso você mantenha o valor da propriedade default do objeto WebActionItem1 como True, a mesma resposta será gerada quando você digitar o seguinte endereço no seu browser:

```
http://localhost/WebCGITeste01.exe
```

ou, no caso do meu provedor de testes:

```
http://www.marceloleao.com.br/cgi-bin/WebCGITeste01.exe
```

Conforme indicado pelo título deste tópico, essa é uma aplicação CGI extremamente simples, mas que ilustrou os seguintes conceitos:

- Toda aplicação CGI precisa de um objeto WebModule.
- As respostas da aplicação são definidas por vários itens incluídos no Editor de Ações.
- Para cada item que define uma ação deve ser definido um procedimento associado ao evento OnAction, e a resposta a ser gerada deve ser um código HTML armazenado na string definida pela propriedade Content do parâmetro Response (que é um objeto da classe TWebResponse).

Em alguns servidores Web de teste, como o que acompanhava o Front Page 98, o endereço deveria ser fornecido como http://localhost/cgi-bin/WebCGITeste01.exe

Exibindo a Data e a Hora do Sistema em uma Página HTML

Neste tópico, aumentaremos um pouco a complexidade do exemplo criado no tópico anterior, de forma a exibir na página gerada pela aplicação CGI a data e a hora do sistema (existindo, neste caso, alguma interação que faça com que a diferencie de uma página completamente estática).

Para exibir a data e a hora do sistema na página HTML gerada em resposta à solicitação do cliente, basta alterar o valor armazenado em Response.Content, redefinindo-o como na linha de código a seguir:

```
Response.Content := '<HTML><BODY><H1>Exemplo de Aplicação CGI</H1><p>Sao exatamente '+Ti5:
meToStr(Time)+' do Dia '+DatetoStr(Date)+'</ BODY ></ HTML > ';
```

A figura a seguir exibe o resultado dessas alterações.

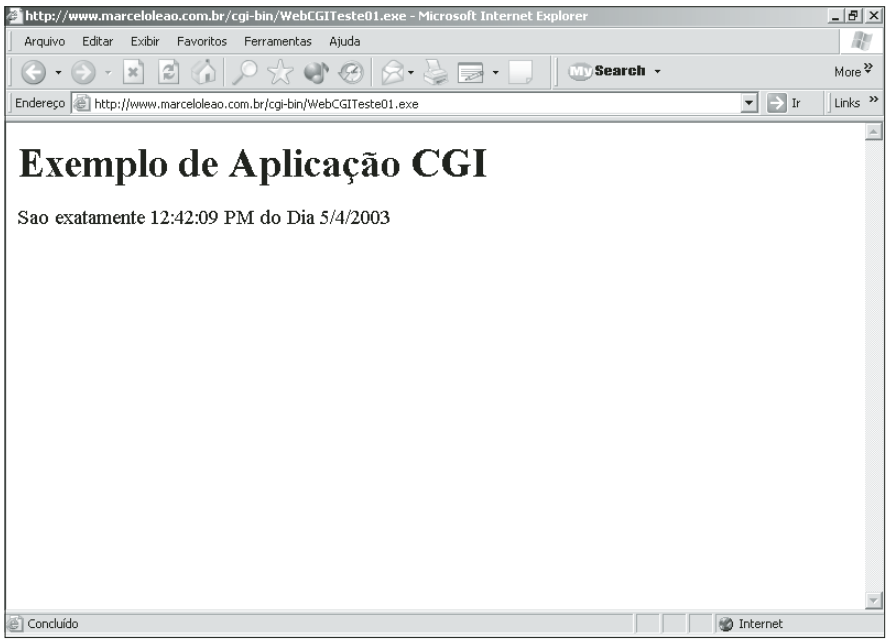


Figura 38.6: Visualizando a data e a hora do sistema no Internet Explorer.

Repare que, nestes casos, como o valor da propriedade default do objeto WebActionItem1 foi definido como True, bastou definir a URL como: http://www.marceloleao.com.br/cgi-bin/WebCGITeste01.exe

(seria http://localhost/ WebCGITeste01.exe no caso de se estar usando o Personal Web Server).

Respondendo a Entrada de Dados de Formulários HTML

Um formulário é definido na linguagem HTML pelas tags <Form> e </Form>, sendo que a primeira tag tem dois parâmetros que representam, respectivamente, uma ação a ser executada e o tipo de método de envio, que pode ser Get ou Post (trabalharemos com o método Get, mas no caso de desenvolvimento de aplicações CGI não faz muita diferença usar Get ou Post, a menos que se trabalhe com informações confidenciais).

Inicialmente, nosso formulário poderia ser representado pelo seguinte código HTML:

```
<Form action = "http://localhost/WebCGITest01.exe" method = "Get"
</Form>
```

É claro que esse código geraria um formulário em branco, o que não teria muito

significado. Para permitir uma entrada de dados pelo cliente, podem-se utilizar algumas caixas de texto que, na linguagem HTML, são definidas pela tag <Input>.

A tag Input, ao ser usada para exibir caixas de texto, deve ser digitada obedecendo à seguinte sintaxe:

```
<Input Type = "Text" Size = "n" Name = "Texto a ser Exibido">
```

Nessa tag, "n" representa o número de caracteres que a caixa de texto deve ser capaz de exibir.

Além disso, a tag <Input> também pode ser usada para exibir botões do tipo Submit e Reset. Nesses casos, a tag <Input> deve ser digitada com a seguinte sintaxe:

```
<Input Type = "Submit" Value = "Texto a Ser Exibido no Botão">
```

e

```
<Input Type = "Reset" Value = "Texto a Ser Exibido no Botão">
```

respectivamente.

Apresentamos a seguir um arquivo bastante simples, que define uma página HTML salva com o nome teste01.htm, e criada usando um editor bastante simples – como o Bloco de Notas do Windows:

```
<HTML>
<BODY>
<H1>Exemplo de Formulário</H1>
<p>
<Form action = "http://www.marceloleao.com.br/cgi-bin/WebCGITeste02.exe" method = "Get"
<p>Nome:
<Input Type = "Text" Size = "32" Name = "Nome">
</p>
<p>Data de Nascimento:
<Input Type = "Text" Size = "20" Name = "Data" >
</p>
<Input Type = "Submit" Value = "Ok">
<Input Type = "Reset" Value = "Limpar">
</Form>
</BODY>
</HTML>
```

Se estiver testando a sua aplicação com o Personal Web Server use

```
<Form action = "http://localhost/WebCGITeste02.exe" method = "Get"
```

A Figura a 37.8 apresenta o aspecto desse formulário elementar no Internet Explorer.

Se digitarmos as expressões "Marcelo Leão" e "28 de Julho de 1966" nas caixas de texto – sem incluir as aspas duplas, evidentemente – e selecionarmos o botão OK do formulário, a seguinte URL será exibida no browser:

```
http://www.marceloleao.com.br/cgi-bin/
WebCGITeste02.exe?Nome=Marcelo+Le%E3o&Data=28+de+julho+de+1966
```

ou, no caso de se estar usando o Personal Web Server:

```
http://localhost/WebCGITeste02.exe?Nome=Marcelo+Le%E3o&Data=28+de+Julho+de+1966
```

Embora seja gerada esta string, a aplicação WebCGITeste02.exe não será executada, pois a mesma ainda não foi criada!

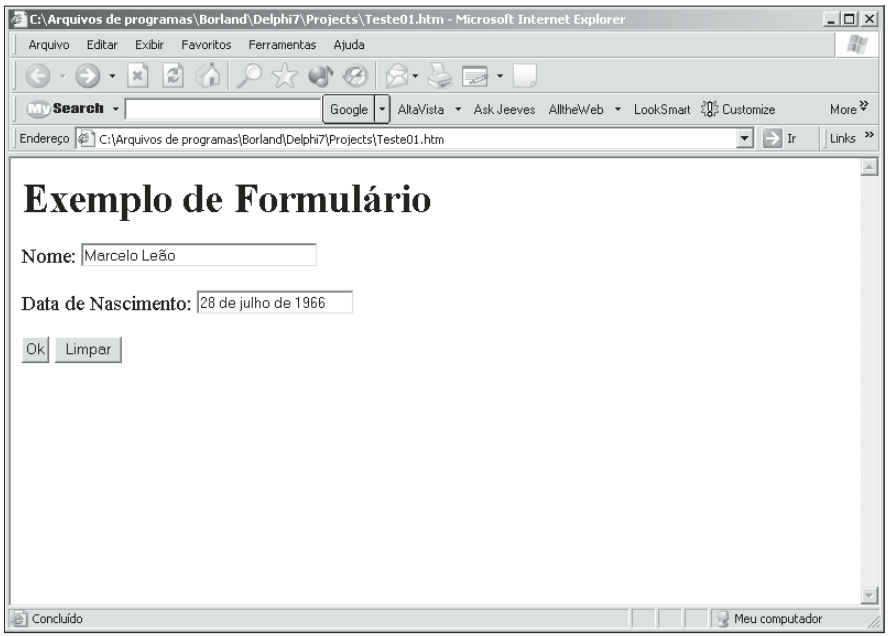


Figura 38.7: Visualizando o formulário no Internet Explorer.

Repare que o endereço da aplicação CGI está separado dos dados enviados pelo formulário por um ponto de interrogação. Após esse ponto de interrogação, os nomes de cada caixa de texto e o valor digitado em cada uma delas são exibidos aos pares, separados por um sinal de igualdade, "=", na forma:

```
Nome_do_Parâmetro=Valor_Digitado
```

Repare que os vários pares de informações são separados por um "e" comercial – & – e que os espaços em branco digitados em cada valor são substituídos por sinais de adição.

Por essa razão, a expressão "28 de Julho de 1966" aparece como

```
"28+de+Julho+de+1966".
```

Quando utilizamos o método Get, cada par de informações "Nome_do_Parâmetro=Valor_Digitado" é armazenado, na forma de uma string, como um elemento de um objeto da classe TStrings (que representa uma lista de strings).

Esse objeto é, na realidade, a propriedade QueryFields do objeto Request, que é, por sua vez, uma propriedade do objeto WebModule.

É fácil verificar, portanto, que ao proceder dessa forma estaremos na realidade manipulando objetos da classe TStrings, juntamente com seus métodos e propriedades.

No caso deste exemplo, os elementos do objeto QueryFields serão:

```
QueryFields[0] : Nome=Marcelo Leão
QueryFields[1] : Data=28 de Julho de 1966
```

Para tornar nossas páginas HTML mais dinâmicas, podemos utilizar o componente PageProducer, apresentado no próximo tópico.

O Componente PageProducer

O Delphi 7 tem um componente chamado PageProducer, que permite que se defina um modelo de página HTML a partir de alguns parâmetros a serem definidos em run-time.

Esse componente tem uma propriedade chamada HTMLDoc, que é um objeto da classe TStrings e que define o código HTML que servirá como modelo da página que será gerada em resposta à solicitação do cliente.

Alternativamente, você pode digitar o modelo de código HTML em um arquivo e definir o nome desse arquivo na propriedade HTMLFile. É importante salientar, no entanto, que essas propriedades são mutuamente excludentes, isto é, definir um valor para uma delas anula o valor definido para a outra.

A principal característica desse modelo de página HTML são as tags <#Nome_da_Tag>, que *não existem na linguagem* HTML e cuja função é única e exclusivamente indicar ao componente PageProducer os parâmetros que serão definidos em run-time.

Para criar uma aplicação CGI que gere uma resposta ao envio de dados pelo formulário (essa aplicação será salva com o nome WebCGITeste02.exe), proceda da seguinte forma:

1. Selecione o item New/Other do menu File do Delphi 7, para exibir a caixa de diálogo New Items.
2. Selecione o item Web Server Application nessa caixa de diálogo.
3. Selecione o botão OK para fechar essa caixa de diálogo. Será exibida a caixa de diálogo New Web Server Application, na qual você deve selecionar a opção CGI Stand-alone executable, como no exemplo anterior.
4. Selecione o botão OK para fechar essa caixa de diálogo e criar um WebModule com o nome default WebModule1.
5. Coloque um componente PageProducer (o segundo componente da página Internet da paleta de componentes) no WebModule1, conforme indicado na Figura 38.8 (esse componente receberá o nome default PageProducer1).
6. Selecione as reticências exibidas à direita da propriedade HTMLDoc do componente PageProducer1. Será exibida a caixa de diálogo String List Editor.
7. Digite nessa caixa de diálogo o modelo de página HTML a ser gerada, conforme mostrado no trecho de código apresentado a seguir:

```
<HTML>
<BODY>
<H1>Olá <#Nome></H1>
<p>
```

```
<H1>Pelas suas informações, você nasceu em <#Data></H1>
<p>
</BODY>
</HTML>
```

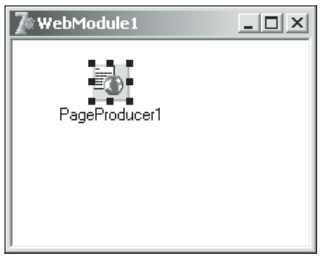


Figura 38.8: Inserindo um componente PageProducer no WebModule.

Esse código define "Nome" e "Data" como parâmetros a serem definidos em run-time, usando o recurso de tags transparentes.

8. Crie um novo item na caixa de diálogo do Editor de Ações do WebModule e mantenha o seu nome default WebActionItem1.
9. Altere o valor da propriedade default desse objeto para True.
10. Defina a propriedade Producer deste objeto como o nome do objeto PageProducer recém-incluído no WebModule.
11. Defina da seguinte maneira o procedimento associado ao evento OnHTMLTag do componente PageProducer1:

```
procedure TWebModule1.PageProducer1HTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
begin
  ReplaceText := Request.QueryFields.Values[TagString];
end;
```

A substituição de parâmetros na página-modelo é feita no procedimento associado ao evento OnHTMLTag, e esse evento é chamado sempre que se quer obter o valor a ser atribuído a um parâmetro (esse parâmetro é passado como a constante TagString no cabeçalho do procedimento). Conforme descrito anteriormente, a propriedade QueryFields é um objeto da classe TStrings que, no caso de utilização do método Get, define uma lista de string em que cada uma delas representa um parâmetro.

Essa lista de strings, como descrito anteriormente, armazena strings cujo conteúdo está na forma Nome=Valor. A classe TStrings, no entanto, tem propriedades que permitem retornar apenas um dos valores, situado de um dos lados do sinal de igual.

A propriedade Values do objeto TStrings, por exemplo, retorna os valores exibidos à direita do sinal de igual. Para obter o valor correspondente à string "Nome", por exemplo, basta digitar essa string entre colchetes. É importante lembrar que esse valor é retornado como uma string, mesmo que os caracteres armazenados sejam todos numéricos.

A propriedade Names, por outro lado, retorna os valores exibidos à esquerda do sinal de igual.

Por essa razão, basta atribuir à variável ReplaceText o valor armazenado na propriedade

Values, correspondente à string TagString.

A figura a seguir apresenta o resultado da execução desse formulário no Internet Explorer.

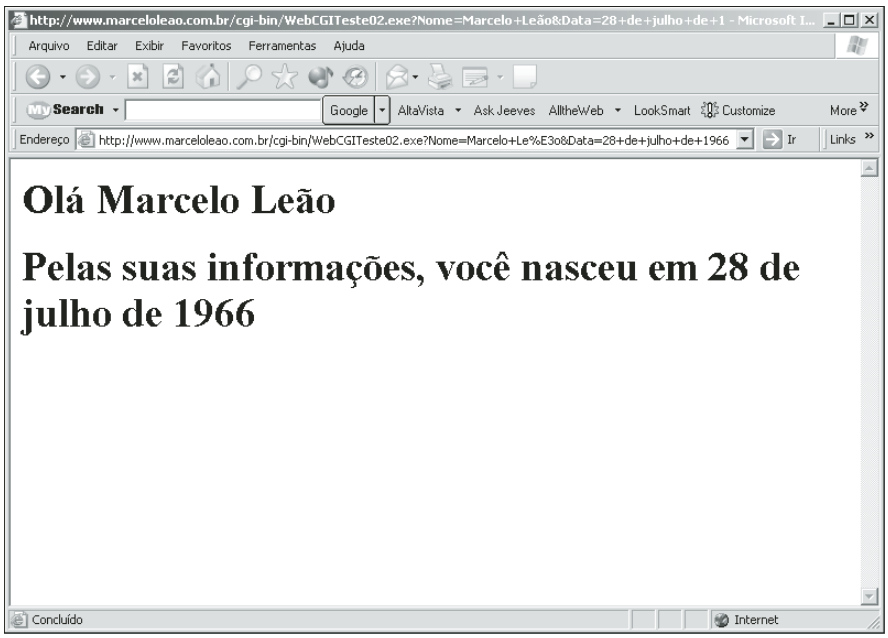


Figura 38.9: Visualizando o resultado no Internet Explorer.

Você já deve ter reparado que as páginas HTML usadas como exemplo neste capítulo são bastante simples. Isso se deve ao fato de que o nosso principal objetivo neste capítulo é apresentar os conceitos relacionados à criação de aplicações CGI com o Delphi 7, e não o de apresentar um tutorial sobre a linguagem HTML.

No próximo tópico, será apresentado o componente DataSetTableProducer, que permite gerar páginas HTML a partir dos valores armazenados nos campos dos registros de uma tabela.

O Componente DataSetTableProducer

O Delphi 7 tem um componente chamado DataSetTableProducer, que permite que se defina um modelo de página HTML que exiba em run-time diversos registros armazenados em uma tabela de um banco de dados.

Esse componente deve trabalhar sempre em conjunto com um componente PageProducer, apresentado anteriormente. Neste tópico, será utilizado o mesmo exemplo apresentado nos tópicos anteriores, fazendo-se, no entanto, algumas alterações de forma a poder utilizar o componente DataSetTableProducer.

Para alterar o exemplo anterior de forma a poder ilustrar a utilização do componente DataSetTableProducer, proceda da seguinte forma:

1. Coloque um componente Table no WebModule (da mesma maneira que se faz em um DataModule) e mantenha o seu nome default Table.

No caso de ter marcado a opção Cross Plataform na criação da aplicação servidora, você não poderá acessar os componentes da página BDE da paleta de componentes. Coloque então um componente ClientDataset no WebModule (ao invés do Table) e mantenha o seu nome default ClientDataset1.

2. Defina os valores das suas propriedade Databasename e TableName para apontar para o arquivo Country.db, geralmente no diretório C:\Arquivos de Programas\Arquivos Comuns\Borland Shared\Data, correspondente ao alias DBDEMOS do BDE.

Caso esteja usando o componente ClientDataset pelas razões já descritas, defina o valor da sua propriedade FileName para apontar para o arquivo Country.cds, geralmente no diretório C:\Arquivos de Programas\Arquivos Comuns\Borland Shared\Data.

3. Defina da seguinte maneira o procedimento associado ao evento OnCreate do WebModule:

```
procedure TWebModule1.WebModuleCreate(Sender: TObject);
begin
   Table1.Open;
end;
```

Caso esteja usando o componente ClientDataset pelas razões já descritas, o código será:

```
procedure TWebModule1.WebModuleCreate(Sender: TObject);
begin
   ClientDataset1.Open;
end;
```

4. Coloque um componente DataSetTableProducer no WebModule e mantenha o seu nome default DataSetTableProducer1.
5. Defina o valor da propriedade DataSet desse componente como ClientDataset1.
6. Ao menos durante esta fase, defina a propriedade Active do componente Table1 como True (e neste caso, ao menos temporariamente, sua propriedade Databasename deverá ser configurada como DBDEMOS).
7. Clique com o botão esquerdo do mouse sobre as reticências exibidas à direita da propriedade Header desse componente. Será exibida a caixa de diálogo String List Editor, na qual deve ser digitado um cabeçalho para a tabela. Digite a expressão "Relação dos países Armazenados na Tabela Country".
8. Clique com o botão esquerdo do mouse sobre as reticências exibidas à direita da propriedade Footer desse componente. Será exibida a caixa de diálogo String List Editor, na qual deve ser digitado um rodapé para a tabela. Digite a expressão "Tabela gerada com o componente DataSetTableProducer".

9. Selecione esse componente com o botão direito do mouse e, no menu suspenso que será exibido, selecione o item Response Editor (você obterá o mesmo resultado selecionando as reticências existentes à direita da propriedade Columns do componente DatasetTableProducer). Será exibida a caixa de diálogo mostrada na Figura 38.10, que permite que se definam:

   - Os campos a serem exibidos.
   - O alinhamento desses campos.
   - A cor de fundo das células que exibem os campos.
   - Uma pré-visualização da tabela gerada.

O alinhamento do texto nas células.

E muitas outras opções. Este é, sem dúvida alguma, um fantástico editor de propriedades!

10. Caso seja necessário, selecione o botão Add All Fields para especificar a utilização de todos os campos.

Repare que os campos ainda aparecem de maneira confusa, como mostrado na Figura 38.10, o que será corrigido nos próximos procedimentos. Este editor permite que os campos da tabela possam ser tratados como objetos independentes.

Você pode deletar um campo (selecionando um botão Delete) e alterar a ordem de exibição desses campos (selecionando os botões Move Up e Move Down).

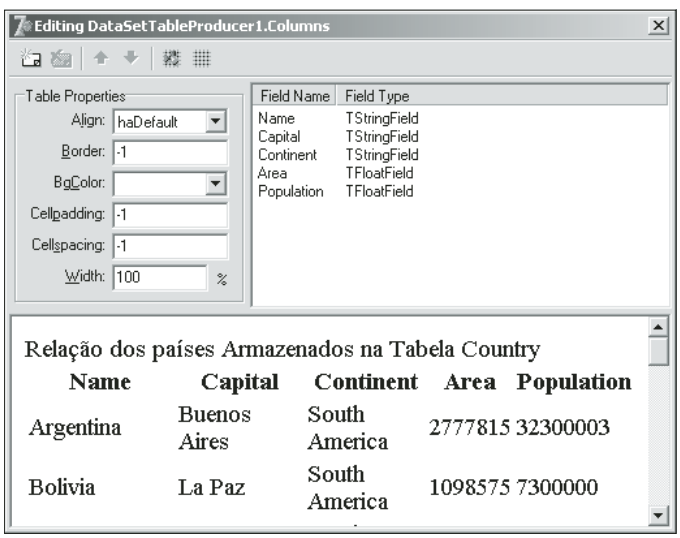


Figura 38.10: O editor de colunas do componente DataSetTableProducer1.

11. Selecione cada um dos campos e altere o valor da sua propriedade Align como haCenter.
12. Altere a propriedade bgColor do GroupBox Table Properties para Gray. O editor de colunas deverá ficar com o aspecto apresentado na Figura 38.11. Repare que a pré-visualização da tabela melhorou bastante.

13. Altere para 1 o valor da propriedade Border do GroupBox Table Properties. O editor de colunas deverá ficar com o aspecto apresentado na Figura 38.12. Repare que a pré-visualização da tabela melhorou ainda mais.

14. Feche o editor de colunas do componente DataSetTableProducer1.

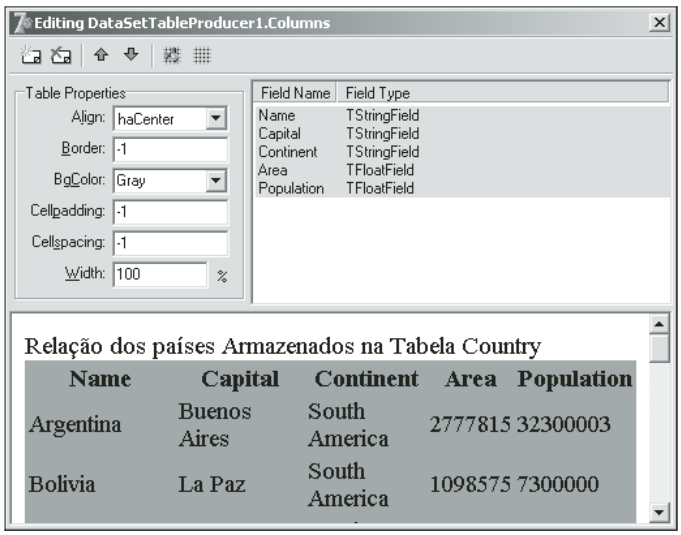


Figura 38.11: O editor de colunas do componente DataSetTableProducer1, após a alteração de algumas propriedades.

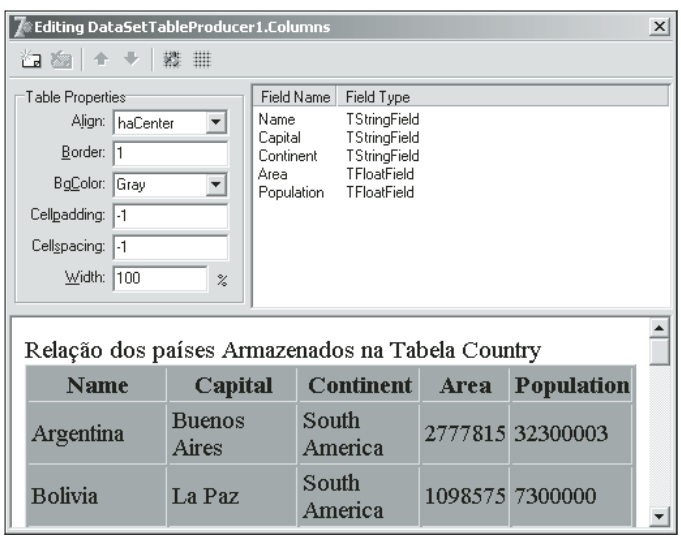


Figura 38.12: O editor de colunas do componente DataSetTableProducer1, após a alteração de algumas propriedades.

15. Crie uma nova página HTML chamada Teste02.htm, com o seguinte código:

```
<HTML>
<BODY>
```

```
<H1>Selecione o Botão Abaixo Para Exibir a Tabela</H1>
    <p>
    <Form action = "http://localhost/WebCGITeste02.exe" method = "GET">
    </p>
    <Input Type = "Submit" Value = "Exibir tabela">
    </Form>
</BODY>
</HTML>
```

Esse é um formulário ainda mais simples, mas sua única função é exibir um botão que, ao ser selecionado pelo usuário, exibe a tabela em uma página HTML.

No meu caso, usei para acessar o provedor de testes:

```
<Form action = "http://www.marceloleao.com.br/cgi-bin//WebCGITeste02.exe" method = "GET">
```

16. Altere a propriedade HTMLDoc do componente PageProducer1 da maneira indicada a seguir. Repare que nesse caso teremos apenas um parâmetro, chamado tabela, que deverá ser substituído pelos vários registros.

```
<HTML>
<BODY>
<H1>Resultado da Solicitação</H1>
 <p>
 <#tabela>
</BODY>
</HTML>
```

17. Redefina da seguinte maneira o procedimento associado ao evento OnHTMLTag do componente PageProducer1:

```
procedure TWebModule1.PageProducer1HTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
begin
  ReplaceText := DataSetTableProducer1.Content;
end;
```

Você também pode excluir o componente PageProducer e definir a propriedade Producer do objeto WebAction1 como sendo igual a DatasetTableProducer1.

18. Visualize a página que contém o formulário no seu browser. Selecione o botão Exibir Tabela para visualizar o resultado, como mostrado nas Figuras 38.13 e 38.14.

Você pode alterar o aspecto da exibição do resultado alterando algumas propriedades do componente DataSetTableProducer. Como um usuário avançado do Delphi 7, você deve explorar ao máximo suas potencialidades.

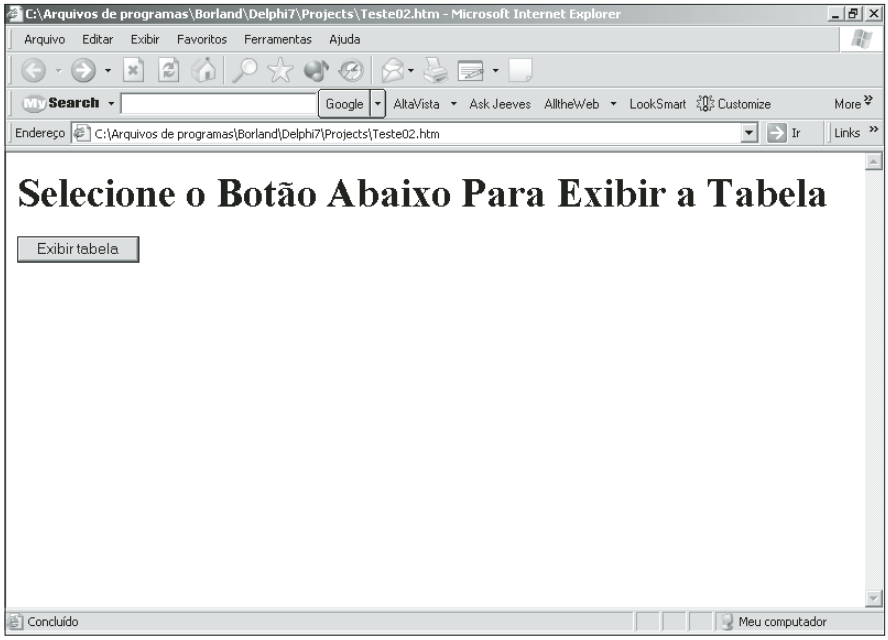


Figura 38.13: Visualizando o formulário no Internet Explorer.

Name	Capital	Continent	Area	Population
Argentina	Buenos Aires	South America	2777815	32300003
Bolivia	La Paz	South America	1098575	7300000
Brazil	Brasilia	South America	8511196	150400000
Canada	Ottawa	North America	9976147	26500000
Chile	Santiago	South America	756943	13200000
Colombia	Bagota	South America	1138907	33000000
Cuba	Havana	North America	114524	10600000
Ecuador	Quito	South America	455502	10600000
El Salvador	San Salvador	North America	20865	5300000
Guyana	Georgetown	South America	214969	800000
Jamaica	Kingston	North America	11424	2500000

Figura 38.14: Visualizando o resultado no Internet Explorer.

É importante lembrar que, neste caso, o mecanismo de acesso deve estar corretamente configurado no servidor, para que o acesso a bancos de dados funcione corretamente. No caso de uma tabela no formato cds, acessada usando o componente ClientDataset, geralmente nenhuma configuração especial é necessária. No caso do acesso via BDE, a propriedade DatabaseName do componente Table deve ser redefinida adequadamente.

No próximo tópico, será apresentado o componente QueryTableProducer, capaz de gerar uma página HTML a partir de uma solicitação do cliente, executando uma declaração SQL. Neste caso, será usado o BDE.

O Componente QueryTableProducer

O Delphi 7 tem um componente chamado QueryTableProducer que permite que se defina um modelo de página HTML que exiba em run-time diversos registros armazenados em uma tabela de um banco de dados, obtidos como resultado de uma consulta SQL parametrizada.

A sua principal vantagem é que, se você definir no formulário HTML um componente cujo Nome seja igual a um parâmetro da declaração SQL definida em um componente Query colocado no WebModule, o valor desse parâmetro será igual ao passado na URL.

Para ilustrar um pouco mais estes conceitos, vamos criar um novo formulário, em um arquivo HTML chamado Teste03.htm, cujo código é apresentado a seguir.

```
<HTML>
<BODY>
<H1>Selecione Uma Opção e o Botão Para Exibir a Tabela</H1>
<p>
<Form action = "http://localhost/WebCGITeste02.exe" method = "GET">
</p>
<Input Type = "Radio" Name = "Continent" Value = "South America" CHECKED>América do Sul
<p>
<Input Type = "Radio" Name = "Continent" Value = "North America">América do Norte
<p>
<Input Type = "Submit" Value = "Exibir tabela">
</Form>
</BODY>
</HTML>
```

No meu caso, usei para acessar o provedor de testes:

```
<Form action = "http://www.marceloleao.com.br/cgi-bin//WebCGITeste02.exe" method = "GET">
```

Nesse formulário, o usuário deverá selecionar um continente para obter informações apenas sobre os países nele situados.

A Figura 38.15 apresenta o formulário gerado.

Quando o usuário selecionar o botão Exibir Tabela, será gerada uma das seguintes URLs, no caso de se estar usando o Personal Web Server:

```
http://localhost/WebCGITeste02.exe?Continent=South+America
```

Ou:

```
http://localhost/WebCGITeste02.exe?Continent=North+America
```

Assim como o componente DataSetTableProducer, o componente QueryTableProducer pode trabalhar em conjunto com um componente PageProducer.

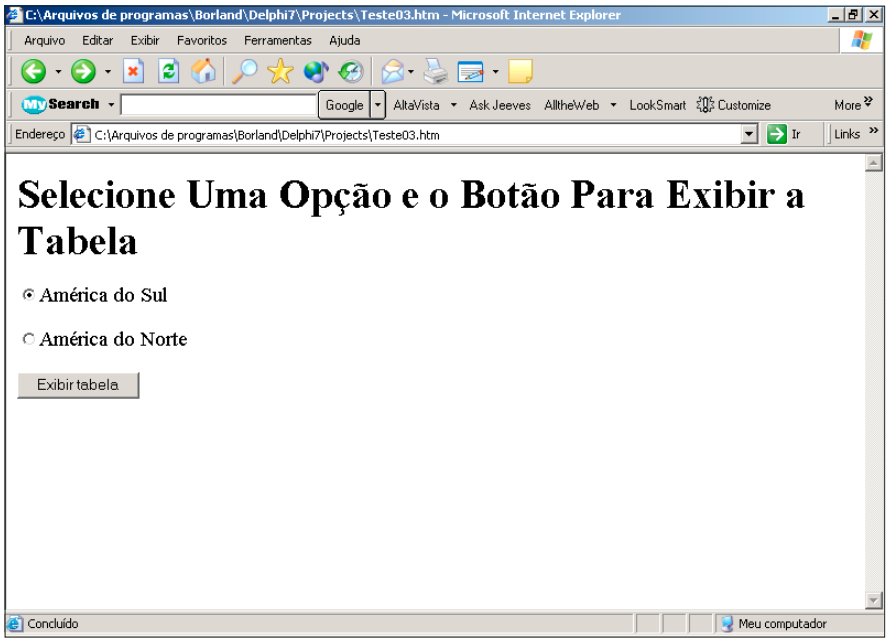


Figura 38.15: Visualizando o resultado no Internet Explorer.

Serão aplicadas algumas alterações no exemplo anterior, de modo a permitir a utilização desse componente.

Para alterar o exemplo anterior, proceda da seguinte forma:

1. Coloque um componente QueyTableProducer no WebModule e mantenha o seu nome default QueryTableProducer1.
2. Coloque um componente Query no WebModule e mantenha o seu nome default Query1.
3. Defina os valores das suas propriedade Databasename e TableName para apontar para o arquivo Country.db, geralmente no diretório C:\Arquivos de Programas\Arquivos Comuns\Borland Shared\Data, correspondente ao alias DBDEMOS do BDE. Ao menos durante esta fase, defina a propriedade Active do componente Table1 como True (e neste caso, ao menos temporariamente, sua propriedade Databasename deverá ser configurada como DBDEMOS).
4. Defina a propriedade SQL do componente Query1 como indicado a seguir.
   ```
   Select * From Country.db Where Continent = :Continent
   ```
5. Selecione as reticências exibidas à direita da propriedade Params do componente Query1 para exibir a janela de configuração de parâmetros, mostrada na Figura 38.16.
6. Selecione o parâmetro Continent e altere as suas propriedades no Object Inspector (defina seu tipo como sendo string).
7. Altere o valor da propriedade Query do componente QueryTableProducer1 para Query1.
8. Altere o valor da Subpropriedade bgColor da propriedade TableAttributes desse componente para Gray.
9. Altere o valor da Subpropriedade Border da propriedade TableAttributes desse componente para 2.

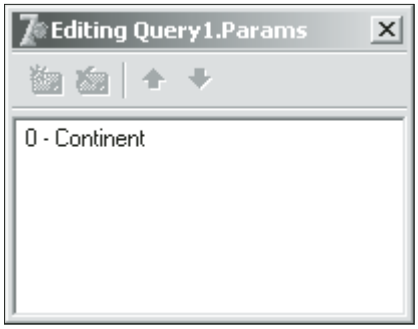


Figura 38.16: Definindo o valor da propriedade Params do componente Query1.

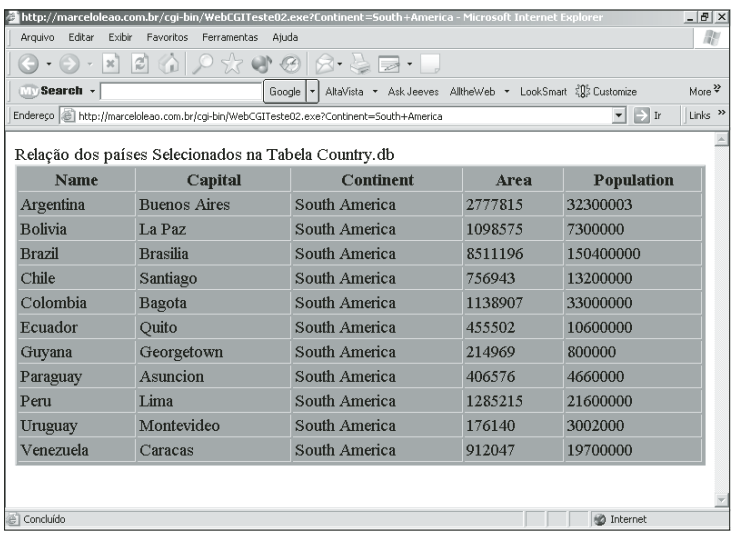
10. Defina a propriedade Header desse componente digitando a expressão "Relação dos países Selecionados na Tabela Country.db" na caixa de diálogo String List Editor.
11. Defina a propriedade Footer desse componente digitando a expressão "Tabela gerada com o componente QueryTableProducer" na caixa de diálogo String List Editor.
12. Redefina da seguinte maneira o procedimento associado ao evento OnHTMLTag do componente PageProducer1:

```
procedure TWebModule1.PageProducer1HTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
begin
  ReplaceText := QueryTableProducer1.Content;;
end;
```

13. Defina da seguinte maneira o procedimento associado ao evento OnCreate do WebModule):

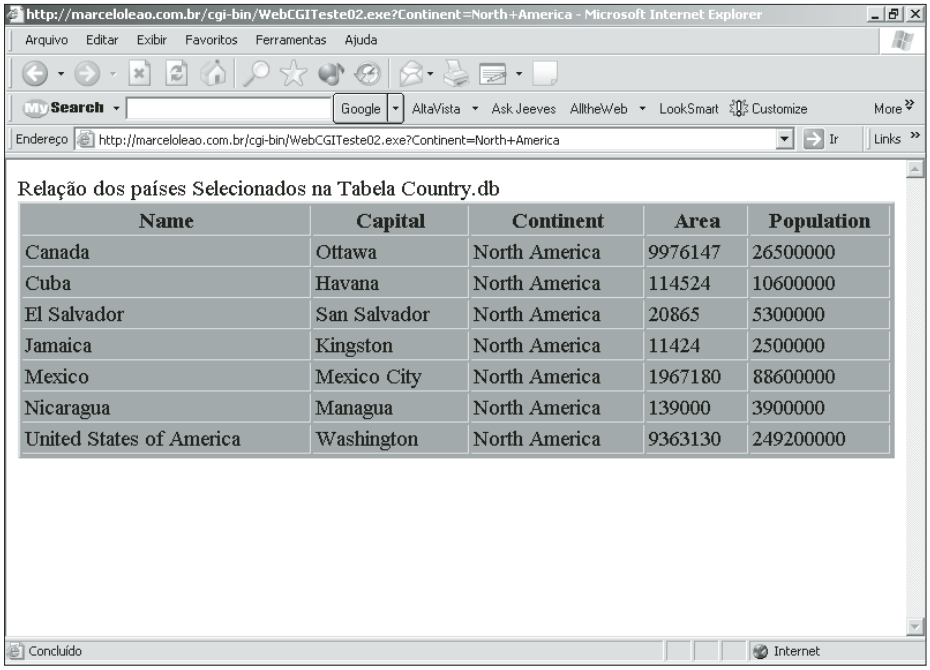
```
procedure TWebModule1.WebModuleCreate(Sender: TObject);
begin
   Query1.Open;
end;
```

14. Salve e recompile a aplicação WebCGITeste02.exe.
15. Selecione uma opção no formulário e o botão Exibir Tabela, para obter os resultados mostrados nas figuras a seguir.



Name	Capital	Continent	Area	Population
Argentina	Buenos Aires	South America	2777815	32300003
Bolivia	La Paz	South America	1098575	7300000
Brazil	Brasilia	South America	8511196	150400000
Chile	Santiago	South America	756943	13200000
Colombia	Bagota	South America	1138907	33000000
Ecuador	Quito	South America	455502	10600000
Guyana	Georgetown	South America	214969	800000
Paraguay	Asuncion	South America	406576	4660000
Peru	Lima	South America	1285215	21600000
Uruguay	Montevideo	South America	176140	3002000
Venezuela	Caracas	South America	912047	19700000

Figura 38.17: Visualizando o resultado no Internet Explorer, após selecionar a opção "América do Sul".



Name	Capital	Continent	Area	Population
Canada	Ottawa	North America	9976147	26500000
Cuba	Havana	North America	114524	10600000
El Salvador	San Salvador	North America	20865	5300000
Jamaica	Kingston	North America	11424	2500000
Mexico	Mexico City	North America	1967180	88600000
Nicaragua	Managua	North America	139000	3900000
United States of America	Washington	North America	9363130	249200000

Figura 38.18: Visualizando o resultado no Internet Explorer, após selecionar a opção "América do Norte".

Repare que, na utilização do componente QueryTableProducer, é importante garantir que os parâmetros da declaração SQL tenham os mesmos nomes dos parâmetros gerados na URL (normalmente definido pela propriedade Name dos componentes de interface do formulário HTML).

KNOW-HOW em: Desenvolvimento de Aplicações CGI com WebSnap

PRÉ-REQUISITOS

- Experiência em programação estruturada com versões anteriores da linguagem Pascal.
- Conhecimentos básicos da linguagem HTML.

METODOLOGIA

- Apresentação do problema: Criação de uma aplicação CGI com a tecnologia WebAnap, utilizando os componentes disponíveis no ambiente de desenvolvimento integrado do Delphi 7.

TÉCNICA

- Apresentação dos procedimentos necessários ao desenvolvimento de aplicações CGI com a tecnologia WebSnap.

A tecnologia Websnap foi incorporada na versão 6 do Delphi e na versão 2 do Kylix, sendo o primeiro passo no desenvolvimento RADS de aplicações para a Internet.

Ao contrário do que ocorre com aplicações CGI desenvolvidas com a tecnologia WebBroker, muitas funcionalidades já estão prontas para serem usadas, mas neste caso o desenvolvedor não tem tanto domínio sobre o código gerado pela aplicação.

Neste exemplo serão apresentadas, através de um exemplo, as principais técnicas para a criação de aplicações Web com a tecnologia WebSnap.

Criando o Módulo Principal de uma Aplicação WebSnap

Para criar o módulo básico de uma aplicação WebSnap, você deve executar os seguintes procedimentos:

1. Selecione File/New/Other no menu do Delphi 7. Será mostrada a caixa de diálogo New Items. Nesta caixa de diálogo, selecione o item WebSnap Application da guia WebSnap, como mostrado na figura a seguir.

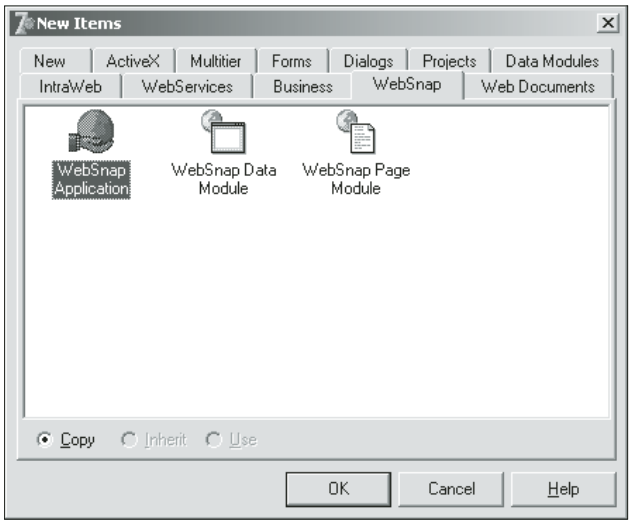


Figura 38.19: A Guia Websnap da caixa de diálogo New Items.

2. Selecione o botão OK. Será exibida a caixa de diálogo New WebSnap Application, mostrada na figura a seguir.

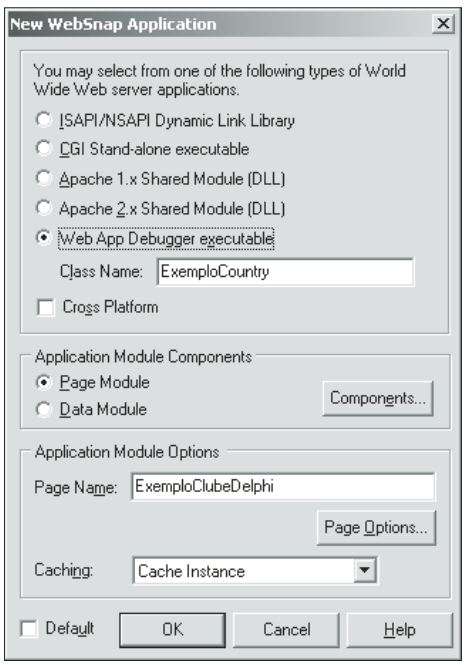


Figura 38.20: A caixa de diálogo New Websnap Application.

Esta caixa de diálogo permite selecionar o tipo de aplicação que será desenvolvida, e inicialmente será selecionada a opção Web App Debugger Executable, que permite testar a aplicação a partir do próprio ambiente de desenvolvimento integrado do Delphi 7. O Web App Debugger é, na realidade, um servidor http integrado, e que associará uma aplicação servidora COM à nossa aplicação.

3. Selecione a opção Web App Debugger Executable, para que se possa depurar a aplicação.
4. Defina CoClassName como ExemploCountry.
5. em Application Module Components selecione a opção Page Module.
6. Em Page Name coloque ExemploClubeDelphi.
7. Se você selecionar o botão Components do GroupBox Application Module Components, verá várias opções disponíveis na caixa de diálogo Web App Components, que inicialmente será mantida com as opções default.
8. Selecione o botão Page Options situado no grupo Application Module Options, e na caixa de diálogo que será exibida defina a opção Producer/Type como AdapterPageProducer, como mostrado na figura a seguir.

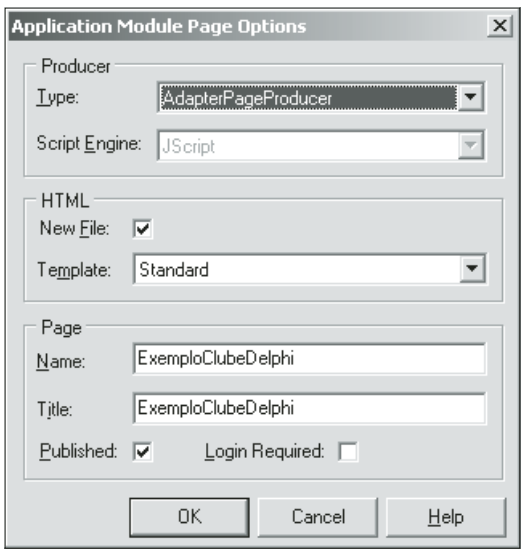


Figura 38.21: A caixa de diálogo Application Module Page Options.

9. Selecione o botão OK para fechar esta caixa de diálogo.
10. Selecione o botão OK para criar o Web Page Module, que será o módulo principal desta nossa aplicação webSnap de exemplo, e reproduzida na Figura 38.22.
11. Selecione File/Save e salve a unit associada com o nome UnitPageModule.pas.
12. Selecione File/Save Project As para salvar a unit principal (até então chamada Unit1) como UnitServer (pois corresponderá ao formulário da aplicação servidora COM) e o projeto com o nome ExemploWebSnap.
13. Inclua um componente session e um componente Table (ambos da página BDE da paleta de componentes) no Web Page Module recém-criado.

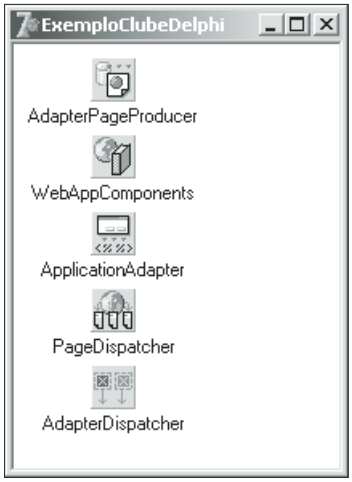


Figura 38.22: O Módulo Principal da Aplicação.

14. Defina o nome do componente Session como SessionCountry.
15. Defina o nome do componente Table como TblCountry.
16. Defina a propriedade AutoSessionName do componente Session como sendo igual a True.
17. Configure as propriedades do componente TblCountry para acessar a tabela Country.db.
18. Defina como True a propriedade Active do componente TblCountry.
19. Usando o Fields Editor do componente TblCountry, adicione objetos para representar todos os campos da tabela, como mostrado na figura a seguir.

Figura 38.23: A janela do Fields Editor.

20. Selecione o campo Name e defina como True a propriedade ProviderFlags/pflnKey do objeto que representa este campo.
21. Adicione um componente DatasetAdapter (página Websnap da paleta de Componentes) ao Web Page Module, e defina a sua propriedade Dataset como TblCountry.

22. Altere a sua propriedade Name para DatasetAdpterCountry.
23. Selecione o componente Application Adapter e altere a sua propriedade Application Title para Exemplo de Aplicação WebSnap.
24. No Editor de códigos, selecione a guia Preview para visualizar a página estática gerada, como mostrado na figura a seguir.

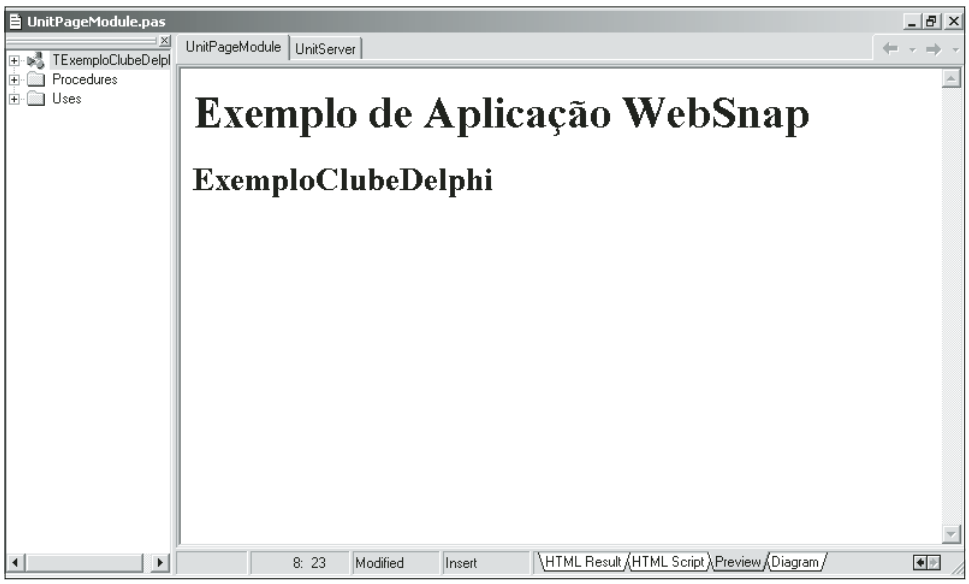


Figura 38.24: Visualizando a página gerada no Editor de Códigos

Cada um dos componentes já existentes na aplicação tem sua própria finalidade.

- O AdapterPageProducer é responsável pela geração final do código HTML a partir das informações obtidas de outros componentes.
- O WebAppComponents centraliza o acesso aos demais componentes.
- O ApplicationAdapter fornece acesso aos campos e ações da variável de script Application. Um exemplo é a propriedade Title desta variável, que é igual à propriedade ApplicationTiltle deste componente.
- O PageDispatcher trata as requisições http dos usuários, redirecionando-as aos responsáveis pelas respostas.
- O AdapterDispatcher é responsável pelo tratamento de informações provenientes de formulários e imagens.
- O DatasetAdapter trata da interface com scripts responsáveis pelas operações com registros provenientes de componentes de acesso a bancos de dados.

Adicionando um Grid Para Exibição dos Registros

Vamos incrementar nossa página, adicionando um Grid para exibição dos registros provenientes da tabela Country.db, acessada através do componente TblCountry.

Para criar este Grid, você deve executar os seguintes procedimentos:

1. Selecione o componente AdapterPageProducer e, no seu menu pop-up, o item Web Page Editor, para exibir a janela mostrada a seguir.

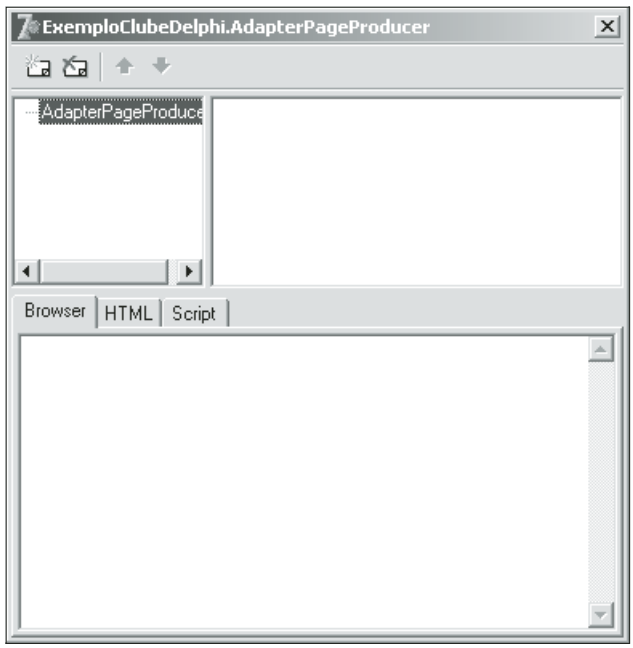


Figura 38.25: A janela do Web Page Editor.

2. No Web Page Editor, selecione o botão New Item e, na caixa de diálogo Add Web Component que será exibida, escolha o item AdapterForm e o botão OK.

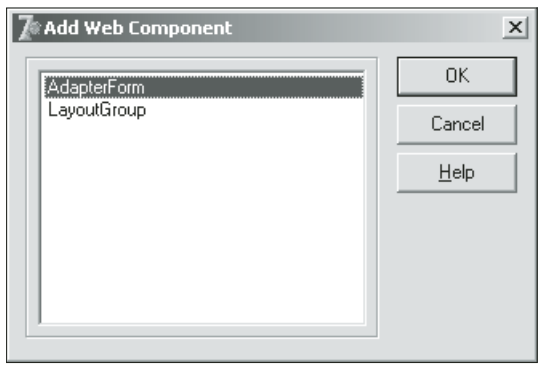


Figura 38.26: A caixa de diálogo Add Web Component.

3. Altere a propriedade Name do objeto AdapterForm1 para CountryForm. Você pode encarar este objeto como um "formulário" no qual serão inseridos "componentes"
4. Selecione o objeto CountryForm e, no seu menu pop-up, o item New Component. Na caixa de diálogo Add Web Component que será exibida, selecione o item AdapterGrid e o botão OK. Considere que você está "inserindo um Grid em um formulário". Repare que o conteúdo da caixa de diálogo Add Web Component varia de acordo com o componente que está selecionado.

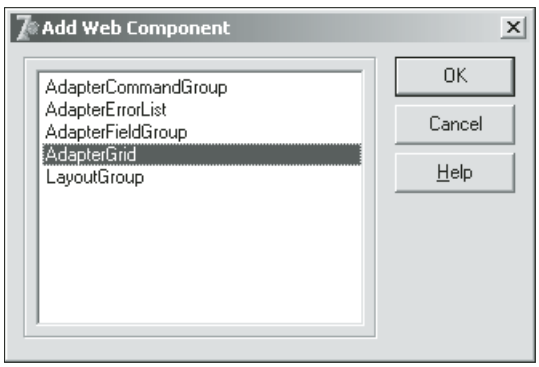


Figura 38.27: A caixa de diálogo Add Web Component.

5. Altere a propriedade Name do objeto AdapterGrid1 para CountryGrid.
6. Defina a propriedade Adapter deste componente como DatasetAdapterCountry (é análogo a definir o Datasource para um DBGrid).

Repare, como mostrado na figura a seguir, que os registros já podem ser visualizados. Repare ainda na hierarquia entre os "componentes".

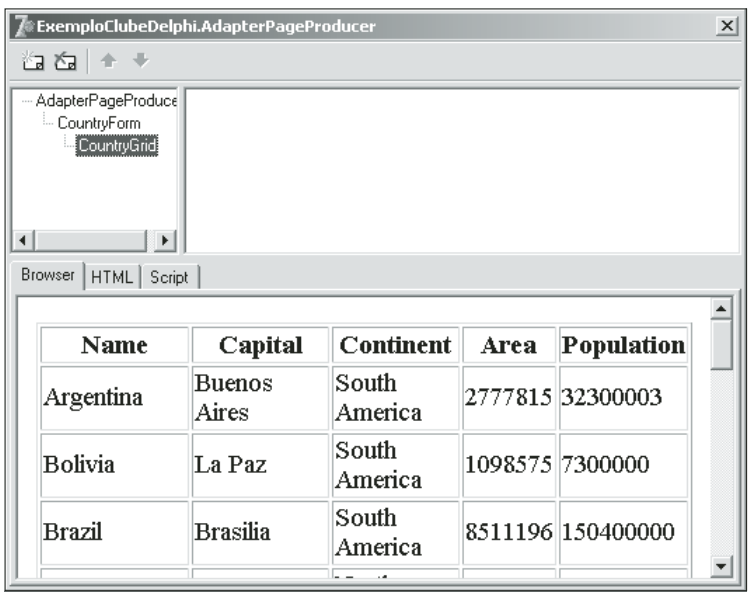


Figura 38.28: Visualizando os Registros da Tabela Country.db.

Adicionando Botões Para a Edição dos Registros do Grid

1. Selecione o item AdapterPageProducer e, no seu menu pop-up, o item WepPageEditor.
2. Selecione o objeto CountryGrid e, no seu menu pop-up, o item Add All Columns.
3. Selecione o objeto CountryGrid e, no seu menu pop-up, o item New Component. Na caixa de diálogo que será exibida, selecione a opção AdapterCommandColumn e o botão OK. Altere a propriedade Name do Objeto criado para GridCommandColumn.
4. Selecione o objeto GridCommandColumn e, no seu menu pop-up, o item Add Commands.
5. Na caixa de diálogo Add Commands, selecione DeleteRow, EditRow e NewRow, como mostrdo na figura a seguir.

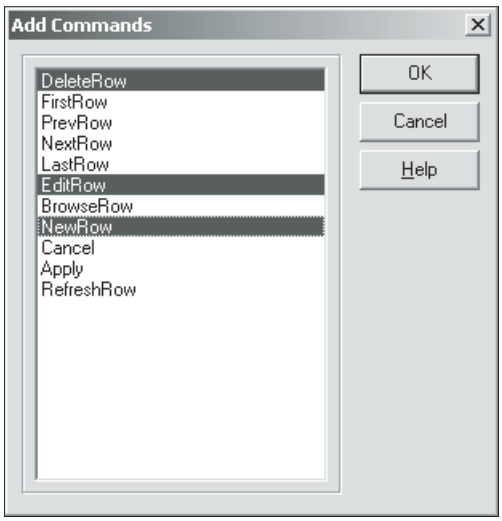


Figura 38.29: A caixa de diálogo Add Commands.

6. Selecione os objetos que representam estes botões e altere as suas propriedades Caption para Excluir, Editar e Novo, respectivamente. Sua página ficará com o aspecto mostrado na figura a seguir.

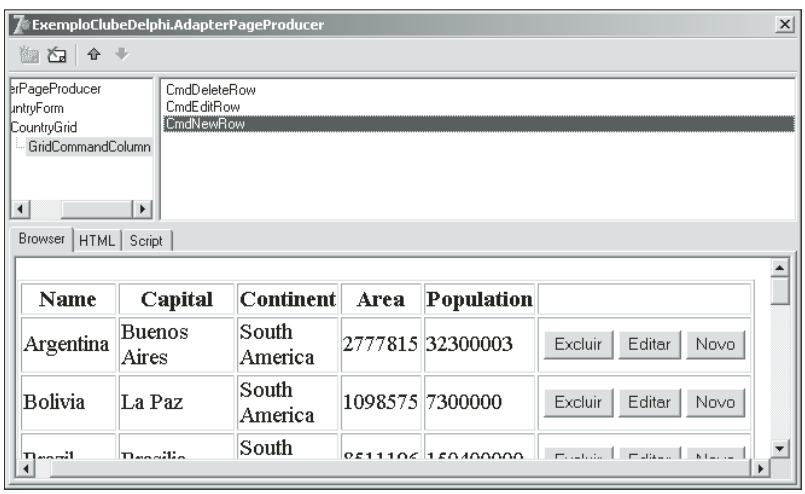


Figura 38.30: Aspecto da página após a inclusão dos botões.

Criando uma Página com um Formulário Para Edição dos Registros

Neste tópico será criado um novo Web Page Module, responsável pela edição dos registros da tabela.

Para criar este Web Page Module, você deve executar os seguintes Procedimentos:

1. Selecione File/New/Other no menu do Delphi 7. Será mostrada a caixa de diálogo New Items. Nesta caixa de diálogo, selecione o item WebSnap Page Module na guia WebSnap e o botão OK.
2. Defina a opção Producer Type como AdapterPageProducer.
3. Defina Page Name como EditaCountry.
4. Desmarque a opção Published.

Sua caixa de diálogo deverá ficar com o aspecto mostrado na figura a seguir:

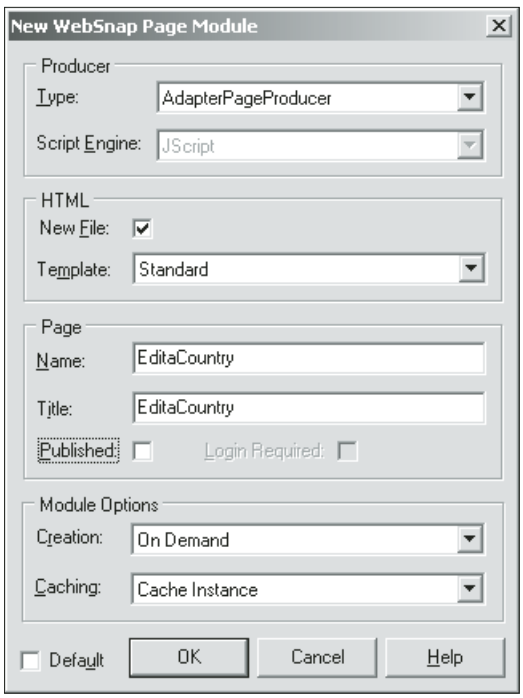


Figura 38.31: Criando o novo Websnap Module.

5. Selecione o botão OK para criar o novo Page Module.
6. Salve a unit associada como UnitPageEditaCountry.
7. Selecione o componente Adapter PageProducer e, no seu menu pop-up, o item WebPageEditor.
8. No Web Page Editor, selecione o botão New Item e, na caixa de diálogo Add Web Component que será exibida, escolha o item AdapterForm e o botão OK.
9. Altere a propriedade Name do objeto AdapterForm1 para EditCountryForm.

10. Selecione o objeto EditCountryForm e, no seu menu pop-up, o item New Component. Na caixa de diálogo Add Web Component que será exibida, selecione o item AdapterFieldGroup e o botão OK.
11. Adicione a unit UnitPageModule à cláusula Uses desta unit, através do menu File/Use unit do Delphi 7.
12. Altere a propriedade Name do objeto AdapterFieldGroup para CountryAdapterFieldGroup.
13. Defina a propriedade Adapter deste componente como ExemploClubeDelphi.DatasetAdapterCountry.
14. Altere a propriedade AdapterMode deste componente para Edit.
15. Selecione o objeto EditCountryForm e, no seu menu pop-up, o item New Component. Na caixa de diálogo Add Web Component que será exibida, selecione o item AdapterCommandGroup e o botão OK.
16. Altere a propriedade Name deste objeto para CountryAdapterCommandGroup.
17. Altere a propriedade DisplayComponent deste objeto para CountryAdapterFieldGroup.
18. Selecione o objeto CountryAdapterCommandGroup e, no seu menu pop-up, o item Add Commands. Na caixa de diálogo que será exibida, selecione Applyncel e RefreshRow, e o botão OK.
19. Altere a propriedade Caption destes botões para Aplicar, Cancelar e Atualizar, respectivamente. sua página ficará com o aspecto mostrado na figura a seguir.

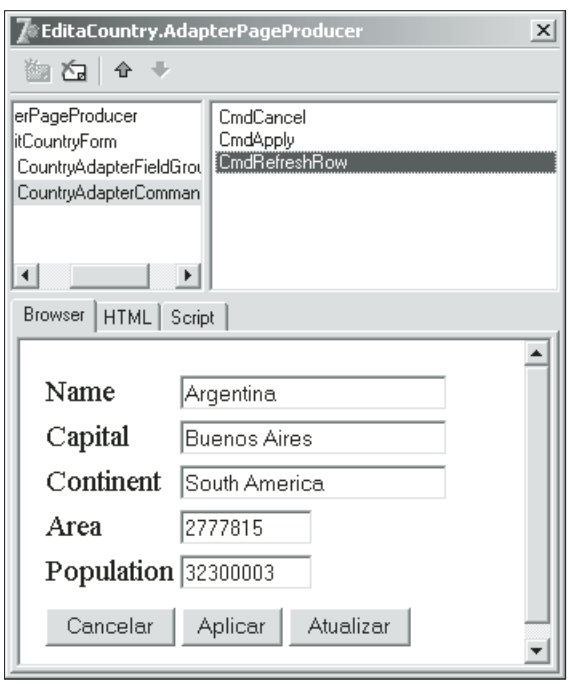


Figura 38.32: Aspecto da página após a inclusão dos botões.

Conectando as Páginas

1. Para conectar as páginas, selecione os objetos que representam os botões Aplicar, Atualizar e Cancelar no Web Page Module EditaCountry, e digite ExemploClubeDelphi na sua propriedade PageName.
2. Selecione os objetos que representam os botões Novo e Editar no Web Page Module EditaExemploClubeDelphi, e digite EditaCountry na sua propriedade PageName.

Testando a Aplicação

Como a aplicação foi criada com a opção Web App Debugger Executable, ela pode ser testada a partir do próprio ambiente do Delphi, bastando para isso executar os seguintes procedimentos:

1. Selecione Run/Run para executar a aplicação servidora, que será representada pelo Form1.
2. Selecione Tools/Web App Debugger. Será exibida a caixa de diálogo mostrada na figura a seguir. Selecione o botão Start e a URL mostrada pata exibir o seu browser default.
3. No browser, selecione a opção correspondente à sua aplicação e o botão " GO".
4. Teste a aplicação.

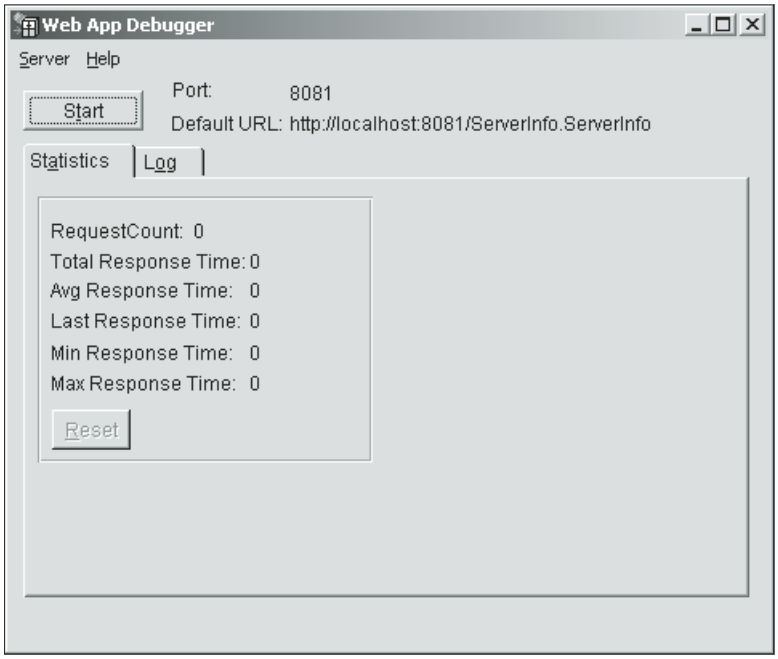


Figura 38.33: O Web App Debugger.

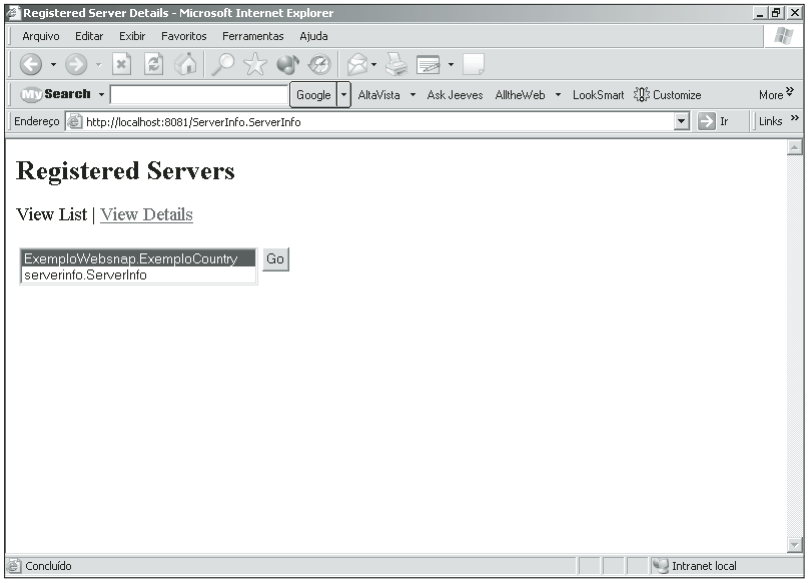


Figura 38.34: Selecionando a aplicação.

As figuras a seguir mostram a aplicação sendo executada no browser.

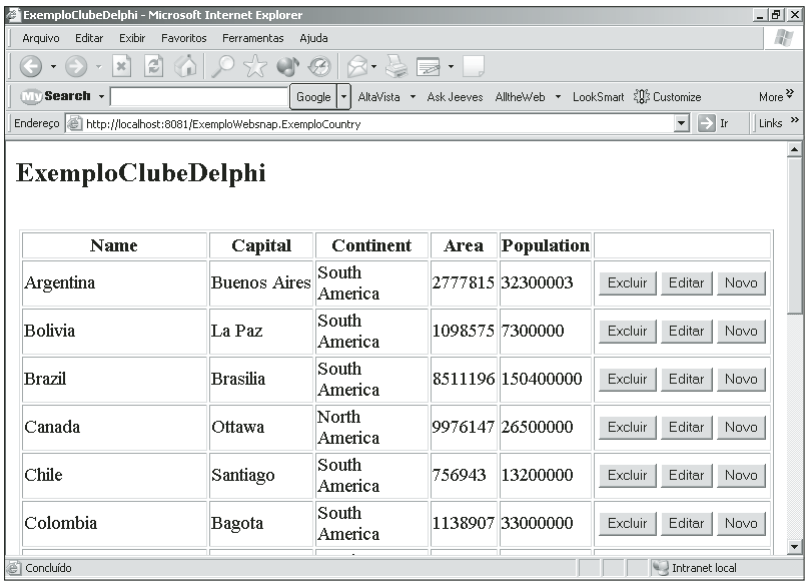


Figura 38.35: Visualizando registros.

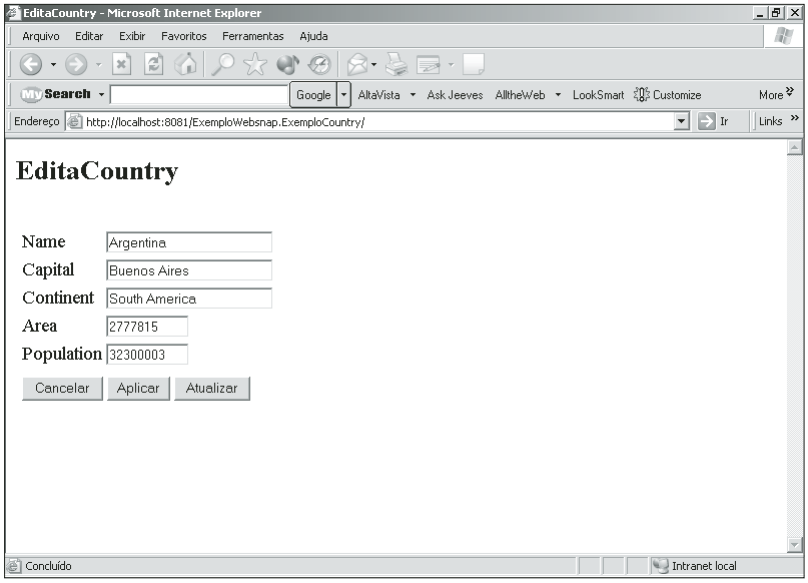


Figura 38.36: Editando registros.

Tratando Erros

Para tratar erros referentes à edição dos registros, você deve executar os seguintes procedimentos:

1. Exiba o Web Page Module secundário EditaCountry.
2. Selecione o componente AdapterPageProducer e, no seu menu pop-up, o item Web Page Editor.
3. Selecione o objeto EditCountryForm e, no seu menu pop-up, o item New Component. Escolha o item AdapterErrorList e o botão OK. Configure a propriedade Adapter deste objeto como sendo igual a ExemploClubeDelphi.DatasetAdapterCountry.
4. Escolha Tools/Debugger Options. Na caixa de diálogo que será exibida, selecione a guia Language Exceptions e desmarque a opção " Stop on Delphi Exceptions".
5. Execute a aplicação. Tente cadastrar um país com um valor inválido para o campo Area, por exemplo, e verifique a mensagem de erro que será exibida na página.

Criando uma Página de Login

Para criar uma página de Login, você deve executar os seguintes procedimentos:

1. Selecione File/New/Other no menu do Delphi 7. Será mostrada a caixa de diálogo New Items. Nesta caixa de diálogo, selecione o itemWebSnap Page Module na guia WebSnap e o botão OK.
2. Defina Producer Type como AdapterPageProducer.
3. Defina Page Name como LoginPage.
4. Selecione o botão OK para criar o novo Page Module.

5. Salve a unit associada como UnitLoginPage.
6. Inclua um componente LoginFormAdapter (página WebSnap da paleta de componentes) neste Websnap Page Module.
7. Altere sua propriedade Name para LoginFormAdapterCountry.
8. Selecione o componente AdapterPageProducer e, no seu menu pop-up, o item Web Page Editor.
9. Selecione o objeto AdapterPageProducer e, no seu menu pop-up, o item New Component. Escolha o item AdapterForm e o botão OK.
10. Altere sua propriedade Name para AdapterFormLogin.
11. Selecione o objeto AdapterFormLogin e, no seu menu pop-up, o item New Component. Escolha o item AdapterFieldGroup e o botão OK.
12. Altere a propriedade Name do objeto criado para AdapterFieldGroupLogin. Configure a sua propriedade Adapter para o LoginFormAdapter recém-criado.
13. Selecione o objeto AdapterFormLogin e, no seu menu pop-up, o item New Component. Na caixa de diálogo que será exibida, selecione o item AdapterCommandGroup e o botão OK.
14. Altere a propriedade Display Component deste objeto AdapterCommandGroup para AdapterFieldGroupLogin.
15. Selecione o objeto AdapterCommandGroup e, no seu menu pop-up, o item New Component. Na caixa de diálogo que será exibida, selecione o item AdapterActionButton e o botão OK.
16. Altere a propriedade ActionName deste objeto AdapterActionButton para Login. Sua página deve ficar com o aspecto mostrado na figura a seguir.

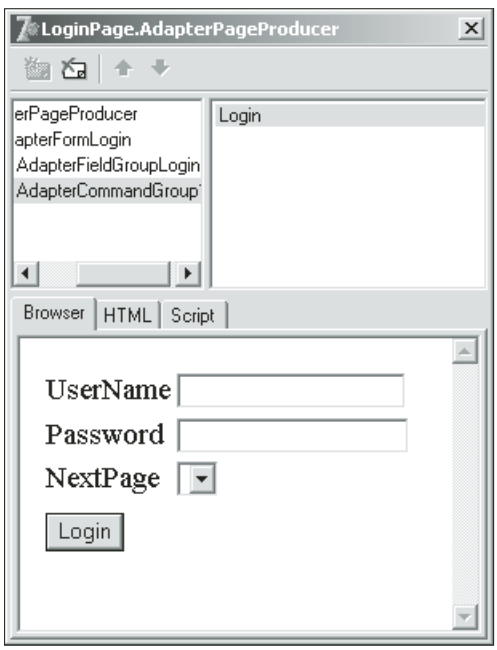


Figura 38.37: Criando a página de Login.

17. Altere a propriedade ActionName deste objeto para Login.
18. Exiba o Módulo Principal da Aplicação (ExemploClubeDelphi).
19. Coloque um componente EndUserSessionAdapter no Web Page Module ExemploClubeDelphi, e altere a sua propriedade Name para EndUserSessionAdapterCountry. Defina sua propriedade LoginPage como LoginPage.
20. Inclua também um componente SessionsService (página WebSnap da paleta de componentes) e altere a sua propriedade Name para SessionsServiceCountry. Agora a página de Login está disponível para ser requisitada por todas as demais páginas desta nossa aplicação WebSnap.

O componente LoginFormAdapter possui uma propriedade chamada PasswordRequired que, se definida como sendo igual a True, não permitirá usuários cuja senha seja nula.

21. Coloque um componente WebUsersList no Websnap Page Module ExemploClube Delphi, e altere a sua propriedade Name para WebUserListCountry.
22. Exiba a caixa de diálogo associada à sua propriedade UserItems e crie diversos usuários, com Username e Password.

Definindo as Páginas que Requerem Login

Falta agora definir as páginas cujo acesso requer a autenticação do usuário. A maneira mais simples de fazer isto consiste em marcar a opção Login Required ao se criar o Web Page Module.

Neste exemplo, em que estes Web Page Modules foram criados sem que tal opção fosse habilitada, devemos retirar os comentários existentes em suas units, que consiste em alterar, por exemplo, a linha de código da seção initialization de cada unit de Initialization

```
initialization
  if WebRequestHandler <> nil then
WebRequestHandler.AddWebModuleFactory(TWebAppPageModuleFactory.Create(TExemploClubeDelphi,
TWebPageInfo.Create([wpPublished {, wpLoginRequired}], '.html'), caCache)
```

Para:

```
initialization
  if WebRequestHandler <> nil then
WebRequestHandler.AddWebModuleFactory(TWebAppPageModuleFactory.Create(TExemploClubeDelphi,
TWebPageInfo.Create([wpPublished, wpLoginRequired], '.html'), caCache)
```

Definindo Direitos de Acesso a Usuários

Cada objeto que representa um usuário na caixa de diálogo associada à propriedade UserItems do componente WebUserList possui uma propriedade chamada AccessRights, que define os seus direitos de acesso.

Para definir estes direitos de acesso na página gerada pelo Web Page Module EditaCountry, faça o seguinte:

1. Selecione o componente AdapterPageProducer deste Web Page Module e, no seu menu pop-up, selecione Web Page Editor.
2. Selecione o objeto CountryAdapterFieldGroup e, no seu menu pop-up, o item Add All Fields, para criar objetos que representam os campos da tabela Country.db e, para cada um destes objetos, defina a sua propriedade ViewMode como vmToggleOnAccess.
3. No Web Page Module principal da aplicação, selecione o componente DatasetAdapterCountry e defina a sua propriedade ModifyAccess como "Modifica".

Apenas usuários cuja propriedade AccessRights for igual a "Modifica" poderão alterar estes campos.

O mesmo vale para Visualização, caso queira que apenas alguns usuários possam visualizar o conteúdo de alguns campos.

O mesmo se aplica ao acesso a uma página. Você pode permitir que apenas usuários com determinado direito de acesso tenham direito a exibir uma página, bastando completar a definição do código exibido na seção initialization de cada unit, como mostrado a seguir para os nossos web page modules:

```
Initialization
  if WebRequestHandler <> nil then
WebRequestHandler.AddWebModuleFactory(TWebAppPageModuleFactory.Create(TExemploClubeDelphi,
TWebPageInfo.Create([wpPublished, wpLoginRequired],
'.html'´´,'ExemploClubeDelphi','Modifica'), caCache)

Initialization
IfWebRequestHandler<>nil then
WebRequestHandler.AddWebModuleFactory(TWebPageModuleFactory.Create(TEditaCountry,
TWebPageInfo.Create([{wpPublished} {wpLoginRequired}],'.html',",",",'Modifica'), crOnDemand,
caCache));
```

KNOW-HOW EM: DESENVOLVIMENTO DE APLICAÇÕES CGI COM INTRAWEB

PRÉ-REQUISITOS

- Experiência em programação estruturada com versões anteriores da linguagem Pascal.

METODOLOGIA

- Apresentação do problema: Criação de uma aplicação ISAPI CGI com a tecnologia Intraweb, utilizando os componentes disponíveis no ambiente de desenvolvimento integrado do Delphi 7.

TÉCNICA

- Apresentação dos procedimentos necessários ao desenvolvimento de aplicações ISAPI CGI com a tecnologia IntraWeb.

A tecnologia IntraWeb consiste em um conjunto de componentes desenvolvidos pela empresa Atozed Software e incorporados pela Borland na versão 7 do Delphi. Isto reforça o fato de que o Delphi não é mais uma linguagem de programação, mas um ambiente de desenvolvimento integrado capaz de incorporar diversas tecnologias, muitas vezes representadas por componentes desenvolvidos por terceiros. O Delphi pode, portanto, ser comparado com um ambiente capaz de integrar várias tecnologias.

Fundamentos da Tecnologia IntraWeb

A tecnologia IntraWeb é representada por diversos componentes, distribuídos nas seguintes páginas da paleta de componentes do Delphi 7:

- IWStandard. Esta página possui versões "Intraweb" dos componentes padrões do Delphi, mas com prefixo IW. O componente IWLabel; por exemplo, é a versão "Intraweb" do Label, o IWEdit, por exemplo, é a versão "Intraweb" do Edit, e assim por diante.
- IWData: Esta página possui versões "Intraweb" dos componentes padrões do Delphi, mas com prefixo IW. O componente IWDBLabel, por exemplo, é a versão "Intraweb" do DBText; o IWDBEdit, por exemplo, é a versão "Intraweb" do DBEdit, e assim por diante.
- IWClientSide: Esta página possui componentes que permitem a visualização de registros de tabelas do lado cliente de uma aplicação Intraweb.
- IWControl: Esta página possui componentes que permitem o acesso a tabelas do lado cliente de uma aplicação Intraweb.

Em nossos exemplos nos concentraremos nos componentes das páginas IWStandard e IWData, que permitem o acesso a dados em um servidor Web.

Serão criadas aplicações do tipo ISAPI DLL, pois aplicações CGI baseadas na Intraweb são, na verdade, serviços a serem disponibilizados pelo servidor, e não simples executáveis que geram códigos HTML, como no caso da tecnologia WebBroker.

Na fase de teste, no entanto, você pode criar uma aplicação do tipo Stand Alone, e usar o item Run do menu Run para testar a sua aplicação.

Criando a Aplicação

A seguir serão apresentados os procedimentos necessários à criação dos formulários para a nossa aplicação Intraweb.

Criando o Formulário Principal da Aplicação

Para iniciar o desenvolvimento desta aplicação, você deve executar os seguintes procedimentos:

1. Selecione File/New/Other, no Delphi 7, para exibir a caixa de diálogo New Items, mostrada na figura a seguir.
2. Nesta caixa de diálogo, selecione a opção ISAPI Application da guia Intraweb e o botão Ok. Será exibida a caixa de diálogo Select Directory mostrada na figura a seguir, na qual você deverá selecionar um diretório para a sua aplicação. No Intraweb, você deve escolher um diretório para cada aplicação, a menos que você esteja convertendo aplicações ISAPI em Stand Alone (e vice-versa).

Caso você esteja querendo testar a aplicação localmente, e não em um servidor Web, selecione a opção Stand Alone Application.

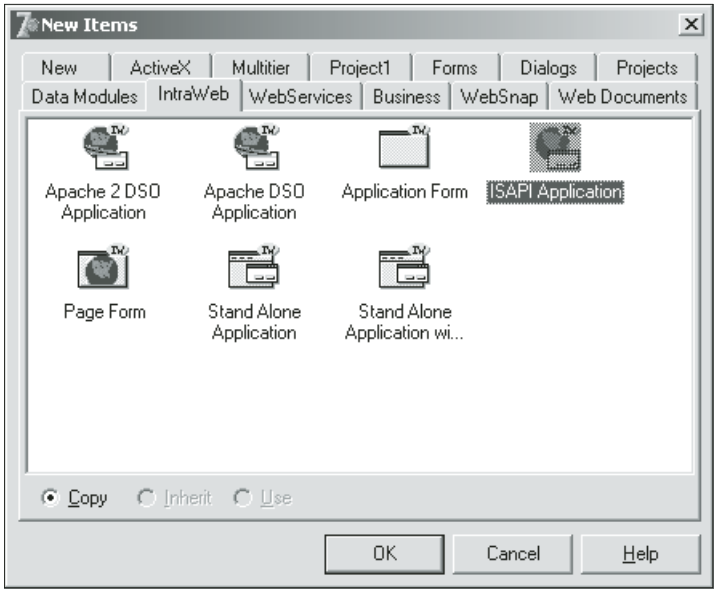


Figura 38.38: A caixa de diálogo New Items.

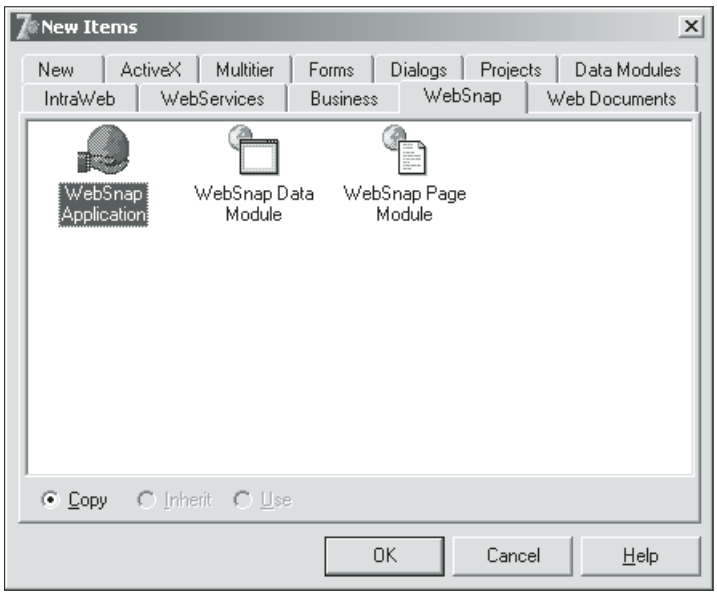


Figura 38.39: A caixa de diálogo Select Directory.

3. Selecione o diretório para a aplicação e o botão Ok. A aplicação será criada e terá dois "formulários Intraweb", denominados formMain e IWServerController. Não interaja com este último formulário – use apenas o formulário principal, inicialmente denominado formMain, e outros que forem criados ao longo do desenvolvimento da aplicação. A figura a seguir mostra o formulário formMain. Repare, a menos na cor inicial, as semelhanças entre este formulário e um formulário de uma aplicação comum em Delphi.

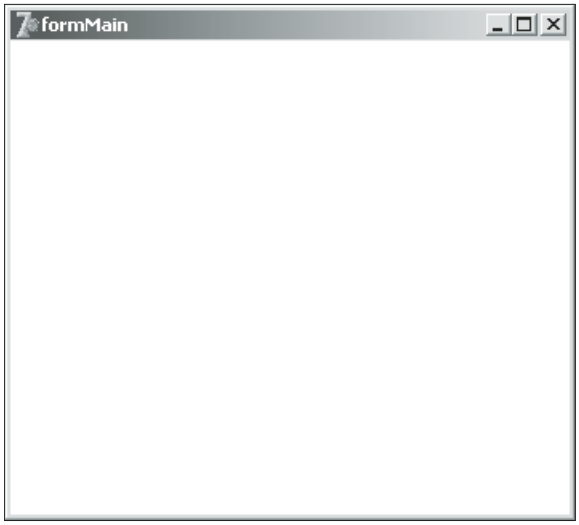


Figura 38.40: O formulário formMain.

4. Altere a propriedade Title do formulário formMain para "Exemplo de Aplicação Intraweb".
5. Salve o projeto com o nome IWCountry.dpr.
6. Inclua um componente Table no formulário formMain e altere o valor da sua propriedade Name para TblCountry.
7. Configure as propriedades Databasename e Tablename deste componente para acessar a tabela Country.db.
8. Defina da seguinte maneira o procedimento associado ao evento OnCreate do formulário formMain:

```
procedure TformMain.IWAppFormCreate(Sender: TObject);
begin
   TblCountry.Open;
end;
```

9. Coloque um componente TMainMenu (página Standard da paleta de componentes) no formulário principal formMain e crie os seguintes menus:

   Menu País, com os itens Cadastrar, Alterar e Excluir.
   Menu Ajuda, com o item Sobre.

10. Coloque um componente IWMenu (página IWStandard da paleta de componentes) e configure a sua propriedade AttachedMenu para MainMenu1 (valor da propriedade Name do componente MainMenu inserido no passo anterior).
11. Inclua um componente Datasource no formulário formMain e defina a sua propriedade Dataset como TblCountry.
12. Inclua um componente IWDBGrid (página IWData da paleta de componentes) no formulário formMain e configure a sua propriedade Datasource como o nome do Datasource incluído no passo anterior.

Caso esteja criando uma aplicação Stand Alone, será possível definir a propriedade Active do componente TblCountry como True, e os registros será visualizados.

13. Defina da seguinte maneira as principais propriedades deste componente:
    - ♦ BGColor: clInfoBk
    - ♦ Datasource: Nome do componente Datasource inserido.
    - ♦ HighLightColor: clYellow.
    - ♦ HighlightRpws: True.
    - ♦ Options/dgIndicator: True.
    - ♦ Options/dgShowTitles: True.
    - ♦ RowClick: False.
14. Coloque um componente IWEditt (página IWStandard da paleta de componentes) e defina sua propriedade Name como IWEDitPesquisa.
15. Inclua um componente IWButton (página IWStandard da paleta de componentes) e defina sua propriedade Name como IWBotaoPesquisa.
16. Defina a propriedade Caption deste componente como "Pesquisa". Reposicione estes componentes para que seu formulário fique com o aspecto mostrado na figura a seguir.

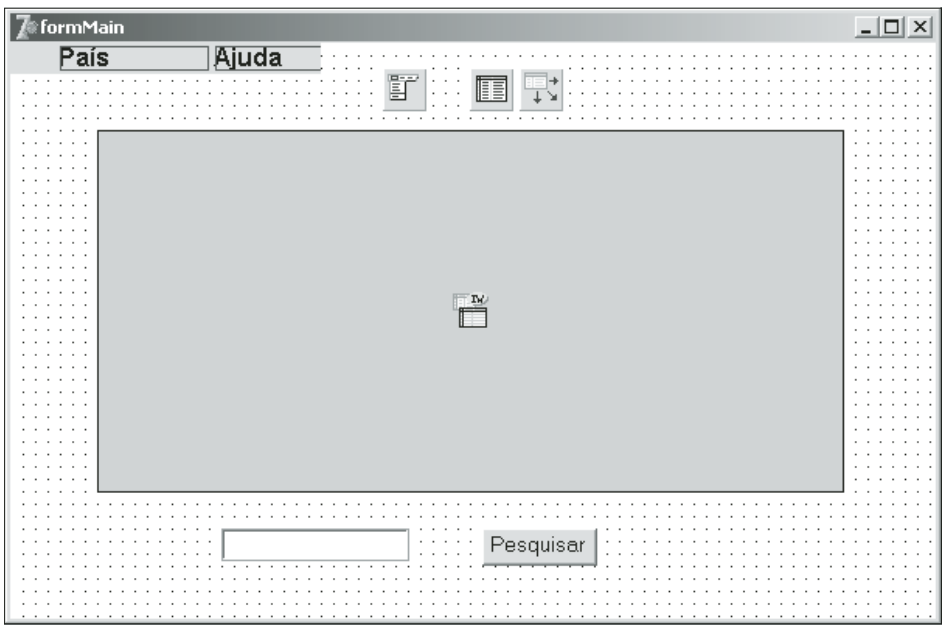


Figura 38.41: Inclusão de componentes de visualização no formulário formMain.

17. Defina da seguinte maneira o procedimento associado ao evento OnClick deste componente, que será usado para pesquisar um país pelo Nome:

```
procedure TformMain.IWBotaoPesquisaClick(Sender: TObject);
begin
   TblCountry.Locate('Name',IWEditPesquisa.Text,[locaseinsensitive,lopartialkey]);
   IWEditPesquisa.Clear;
end;
```

18. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Sobre do menu Ajuda:

```
procedure TformMain.Sobre1Click(Sender: TObject);
begin
    WebApplication.ShowMessage('Exemplo de Aplicação Intraweb!');
end;
```

19. Defina da seguinte maneira o procedimento associado ao evento OnRender deste formulário:

```
procedure TformMain.IWAppFormRender(Sender: TObject);
begin
   tblcountry.Refresh;
end;
```

20. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Cadastrar do menu Países:

```
procedure TformMain.Cadastrar1Click(Sender: TObject);
var
  FormCadastro : TFormCadastro;
begin
    FormCadastro := TFormCadastro.Create(WebApplication);
    FormCadastro.Show;
end;]
```

Se tentar compilar a aplicação neste momento será exibida uma mensagem de erro, pois o formulário de cadastro ainda não foi criado. Repare ainda que o formulário será criado durante a execução da aplicação, mediante uma chamada ao método construtor da classe correspondente.

21. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Alterar do menu Países:

```
procedure TformMain.Alterar1Click(Sender: TObject);
var
  FormAltera : TFormAltera;
begin
    FormAltera := TFormAltera.Create(WebApplication);
    FormAltera.Show;
end;
```

22. Defina da seguinte maneira o procedimento associado ao evento OnClick do item Excluir do menu Países:

```
procedure TformMain.Excluir1Click(Sender: TObject);
var
  FormExclusao : TFormExclusao;
begin
    FormExclusao := TFormExclusao.Create(WebApplication);
    FormExclusao.Show;
    TblCountry.Refresh;
end;
```

Criando o Formulário de Cadastro

Para criar o formulário de cadastro, você deve executar os seguintes procedimentos:

1. Selecione File/New/Other, no Delphi 7, para exibir a caixa de diálogo New Items.
2. Nesta caixa de diálogo, selecione a opção Application Form da guia Intraweb e o botão Ok. Será criado um formulário em branco.

3. Altere as propriedades Name e Title deste formulário para FormCadastro e "Cadastro de Países", respectivamente.
4. Salve a unit associada a este formulário com o nome UnitCadastro.pas.
5. Selecione File/Use Unit para incluir o nome da unit correspondente ao formulário principal na cláusula uses desta unit.
6. Inclua cinco componentes IWLabel (página IWStandard da paleta de componentes) e defina suas propriedades Caption como Name, Capital, Continent, Population e Área. Posicione estes componentes verticalmente do lado esquerdo do formulário.
7. Inclua um componente Datasource no formulário formMain e defina a sua propriedade Dataset como formMain.TblCountry.
8. Inclua cinco componentes IWDBEdit (página IWData da paleta de componentes) e defina a propriedade Datasource destes componentes como o componente Datasource inserido neste formulário. Coloque um componente IWEdit ao lado de cada um dos componentes IWLabel inseridos no passo anterior e defina a propriedade Datafield de cada um deles como sendo o nome do campo que será exibido (e que será igual à propriedade Caption do componente IWLabel correspondente). Caso o nome do campo não seja exibido (porque ao conexão não está ativa) digite explicitamente o nome do campo.
9. Defina da seguinte maneira o procedimento associado ao evento OnCreate deste formulário:

```
procedure TFormCadastro.IWAppFormCreate(Sender: TObject);
begin
    Datasource1.DataSet.Append;
end;
```

10. Inclua um componente IWButton (página IWStandard da paleta de componentes) e defina sua propriedade Name como IWBotaoCadastrar.
11. Defina a propriedade Caption deste componente como "Cadastrar".
12. Defina da seguinte maneira o procedimento associado ao evento OnClick deste componente:

```
procedure TFormCadastro.IWBotaoCadastrarClick(Sender: TObject);
begin
    Datasource1.DataSet.Post;
    Datasource1.DataSet.Append;
end;
```

13. Inclua outro componente IWButton (página IWStandard da paleta de componentes) e defina sua propriedade Name como IWBotaoVoltar.
14. Defina a propriedade Caption deste componente como "Voltar".
15. Defina da seguinte maneira o procedimento associado ao evento OnClick deste componente:

```
procedure TFormCadastro.IWBotaoVoltarClick(Sender: TObject);
begin
  Hide;
  Free;
end;
```

Repare que inicialmente se oculta o formulário com o método Hide (fazendo com que o formulário principal volte a ser exibido) e a sua memória é liberada com o método Free.

Reposicione estes componentes para que seu formulário fique com o aspecto mostrado na figura a seguir.

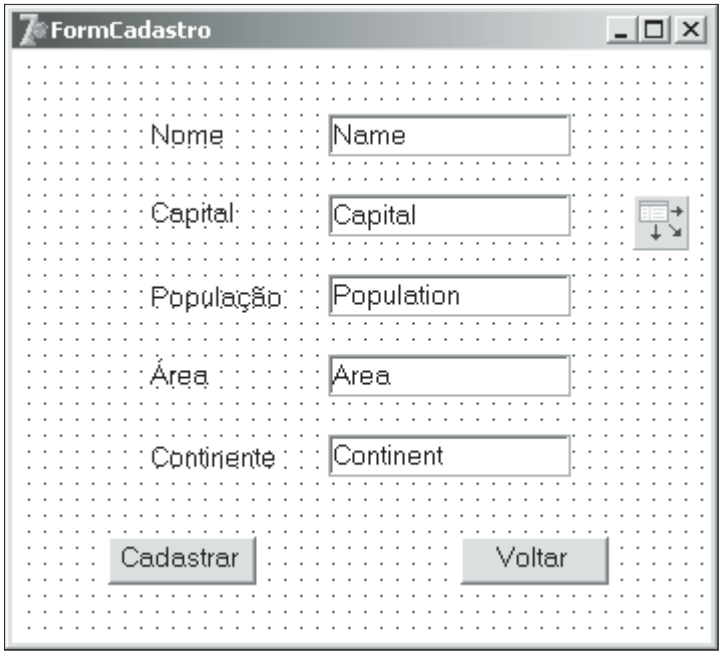


Figura 38.42: Inclusão de componentes de visualização no formulário formCadastro.

16. No formulário principal, selecione File/Use Unit para incluir o nome da unit correspondente ao formulário de cadastro na cláusula uses da unit correspondente ao formulário principal.

Criando o Formulário de Alteração

Para criar o formulário de alteração, você deve executar os seguintes procedimentos:

1. Selecione File/New/Other, no Delphi 7, para exibir a caixa de diálogo New Items.
2. Nesta caixa de diálogo, selecione a opção Application Form da guia Intraweb e o botão Ok. Será criado um formulário em branco.
3. Altere as propriedades Name e Title deste formulário para FormAltera e "Alteração de Países", respectivamente.
4. Salve a unit associada a este formulário com o nome UnitAltera.pas.
5. Selecione File/Use Unit para incluir o nome da unit correspondente ao formulário principal na cláusula uses desta unit.
6. Inclua cinco componentes IWLabel (página IWStandard da paleta de componentes) e defina suas propriedades Caption como Name, Capital, Continent, Population e Área. Posicione estes componentes verticalmente do lado esquerdo do formulário.
7. Inclua um componente Datasource no formulário formMain e defina a sua propriedade Dataset como formMain.TblCountry.
8. Inclua cinco componentes IWDBEdit (página IWData da paleta de componentes) e defina a propriedade Datasource destes componentes como o componente Datasource inserido neste formulário. Coloque um componente IWEdit ao lado de cada um dos componentes IWLabel

inseridos no passo anterior e defina a propriedade Datafield de cada um deles como sendo o nome do campo que será exibido (e que será igual à propriedade Caption do componente IWLabel correspondente). Caso o nome do campo não seja exibido (porque ao conexão não está ativa) digite explicitamente o nome do campo.

9. Inclua outro componente IWButton (página IWStandard da paleta de componentes) e defina sua propriedade Name como IWBotaoVoltar.
10. Defina a propriedade Caption deste componente como "Voltar".
11. Defina da seguinte maneira o procedimento associado ao evento OnClick deste componente:

```
procedure TFormAltera.IWBoiaoVoltarClick(Sender: TObject);
begin
 try
  Datasource1.DataSet.post;
 except
 end;
  Hide;
  Free;

end;
```

Reposicione estes componentes para que seu formulário fique com o aspecto mostrado na figura a seguir.

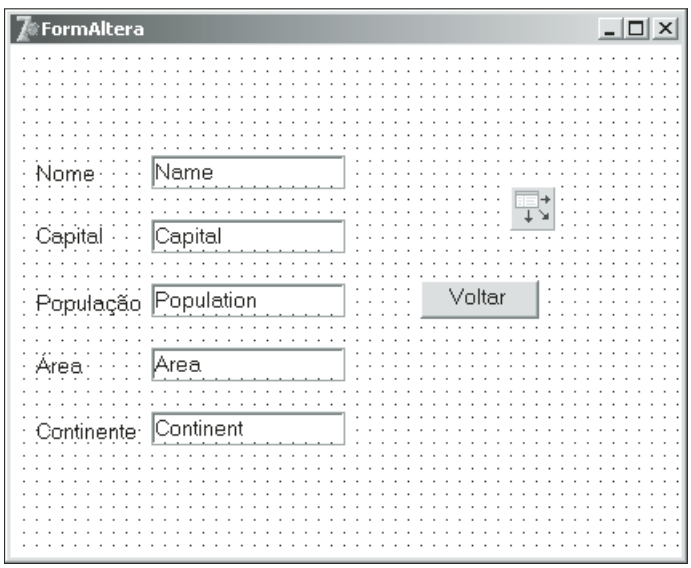


Figura 38.43: Inclusão de componentes de visualização no formulário formAltera.

12. No formulário principal, Selecione File/Use Unit para incluir o nome da unit correspondente ao formulário de alteração na cláusula uses da unit correspondente ao formulário principal.

Criando o Formulário de Exclusão

Para criar o formulário de exclusão, você deve executar os seguintes procedimentos:

1. Selecione File/New/Other, no Delphi 7, para exibir a caixa de diálogo New Items.

2. Nesta caixa de diálogo, selecione a opção Application Form da guia Intraweb e o botão Ok. Será criado um formulário em branco.
3. Altere as propriedades Name e Title deste formulário para FormExclusao e "Confirmação de Exclusão", respectivamente.
4. Salve a unit associada a este formulário com o nome UnitExclusao.pas.
5. Selecione File/Use Unit para incluir o nome da unit correspondente ao formulário principal na cláusula uses desta unit.
6. Inclua um componente IWLabel (página IWStandard da paleta de componentes) e defina sua propriedade Caption como "Confirma a Exclusão do Registro?". Posicione este componente no topo do formulário.
7. Inclua cinco componentes IWLabel (página IWStandard da paleta de componentes) e defina suas propriedades Caption como Name, Capital, Continent, Population e Área. Posicione estes componentes verticalmente do lado esquerdo do formulário.
8. Inclua um componente Datasource no formulário formMain e defina a sua propriedade Dataset como formMain.TblCountry.
9. Inclua cinco componente, IWDBEdit (página IWData da paleta de componentes) e defina a propriedade Datasource destes componentes como o componente Datasource inserido neste formulário. Coloque um componente IWEdit ao lado de cada um dos componentes IWLabel inseridos no passo anterior e defina a propriedade Datafield de cada um deles como sendo o nome do campo que será exibido (e que será igual à propriedade Caption do componente IWLabel correspondente). Caso o nome do campo não seja exibido (porque ao conexão não está ativa) digite explicitamente o nome do campo.
10. Inclua um componente IWButton (página IWStandard da paleta de componentes) e defina sua propriedade Name como IWBotaoExcluir.
11. Defina a propriedade Caption deste componente como "Sim".
12. Defina da seguinte maneira o procedimento associado ao evento OnClick deste componente:

```
procedure TFormExclusao.IWBotaoExcluirClick(Sender: TObject);
begin
    Datasource1.DataSet.Delete;
    Hide;
    Free;
end;
```

13. Inclua outro componente IWButton (página IWStandard da paleta de componentes) e defina sua propriedade Name como IWBotaoCancelar.
14. Defina a propriedade Caption deste componente como "Não".
15. Defina da seguinte maneira o procedimento associado ao evento OnClick deste componente:

```
procedure TFormExclusao.IWBotaoCancelarClick(Sender: TObject);
begin
    Hide;
    Free;
end;
```

Repare que os dois botões ocultam o formulário e liberam sua memória, mas apenas um deles exclui o registro.

Reposicione estes componentes para que seu formulário fique com o aspecto mostrado na figura a seguir.

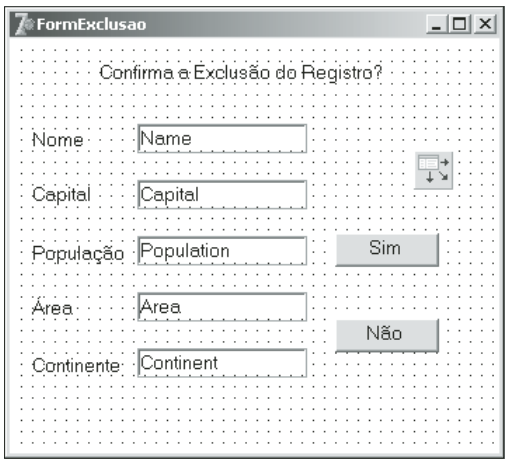


Figura 38.44: Inclusão de componentes de visualização no formulário formCadastro.

16. No formulário principal, Selecione File/Use Unit para incluir o nome da unit correspondente ao formulário de exclusão na cláusula uses da unit correspondente ao formulário principal.

Testando a Aplicação

Se você tiver criado uma aplicação do tipo ISAPI, basta seguir as instruções do seu provedor para fazer o upload dos arquivos necessários e executar a aplicação a partir de um browser, como mostrado nas figuras a seguir.

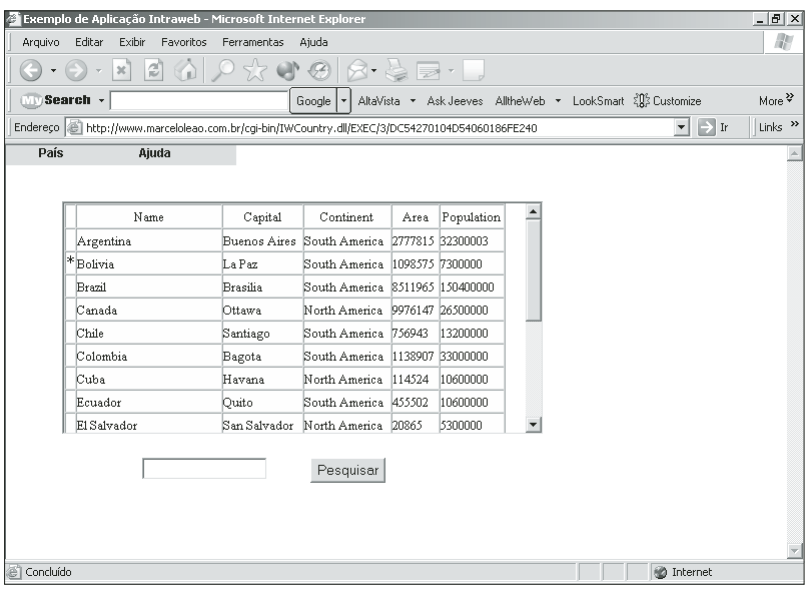


Figura 38.45: Visualizando o Formulário Principal.

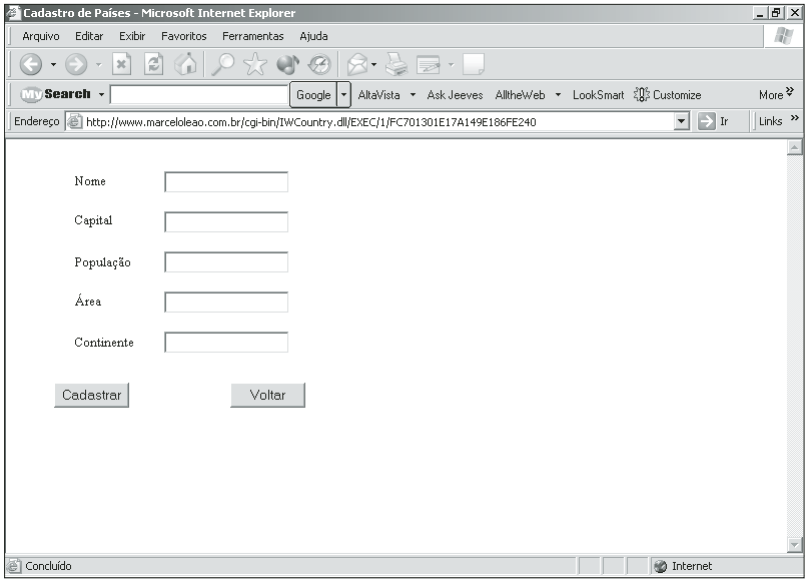


Figura 38.46: Visualizando o Formulário de Cadastro.

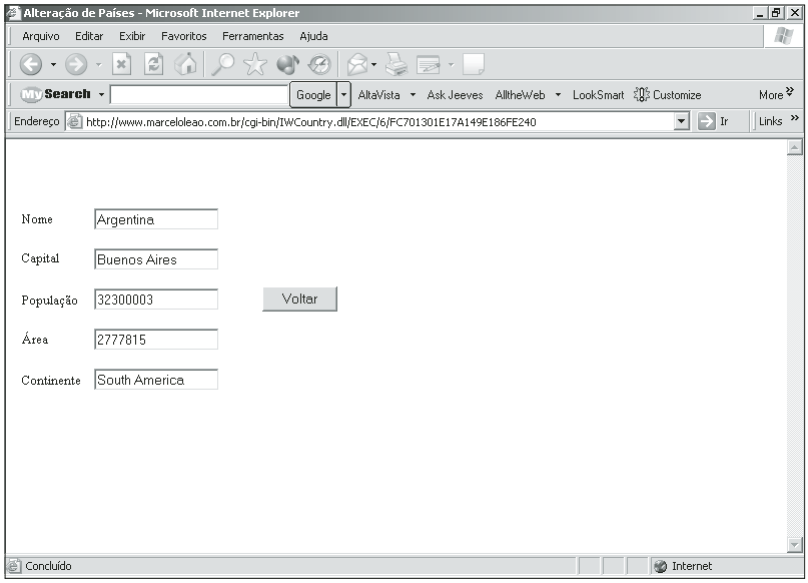


Figura 38.47: Visualizando o Formulário de Alteração.

Caso você esteja testando localmente uma aplicação Stand Alone, pode selecionar o item Run do menu Run para exibir a janela do My Intraweb Application Server e, a partir desta janela, o primeiro botão da sua barra de botões. Repare o endereço 127.0.0.1 na barra de endereços do browser.

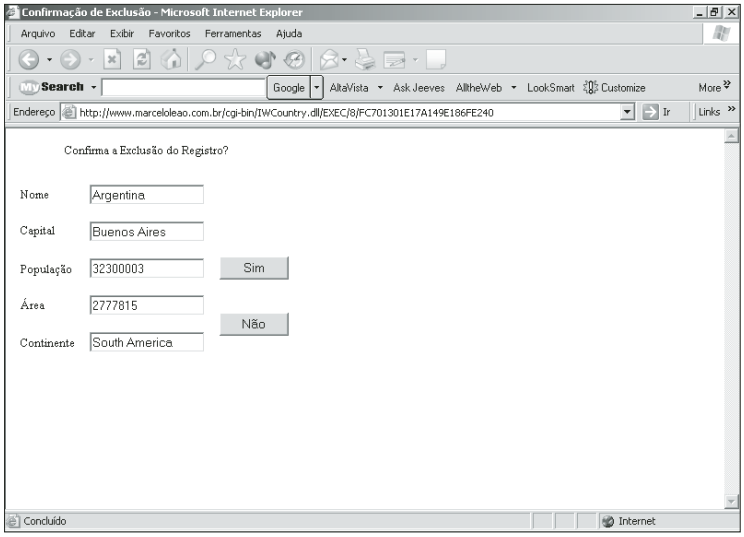


Figura 38.48: Visualizando o Formulário de Exclusão.

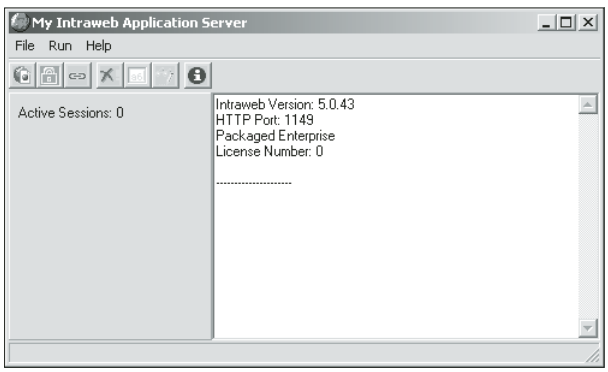


Figura 38.49: A janela do My Intraweb Application Server.

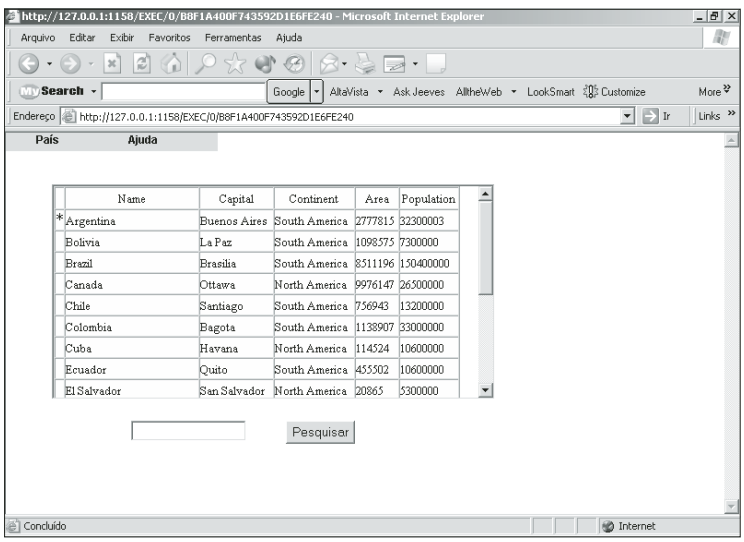
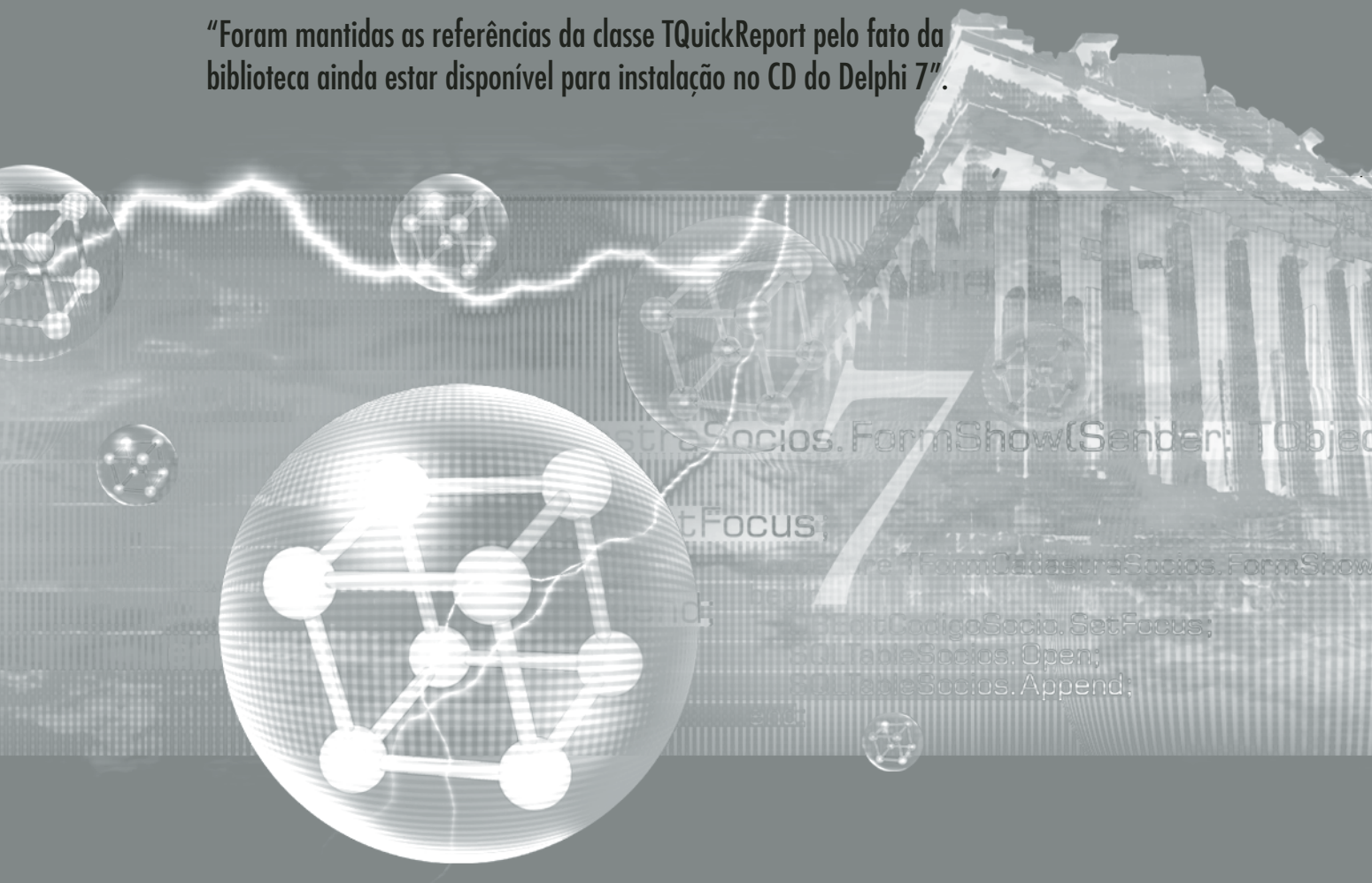


Figura 38.50: Executando Localmente a aplicação Stand Alone.

Parte III

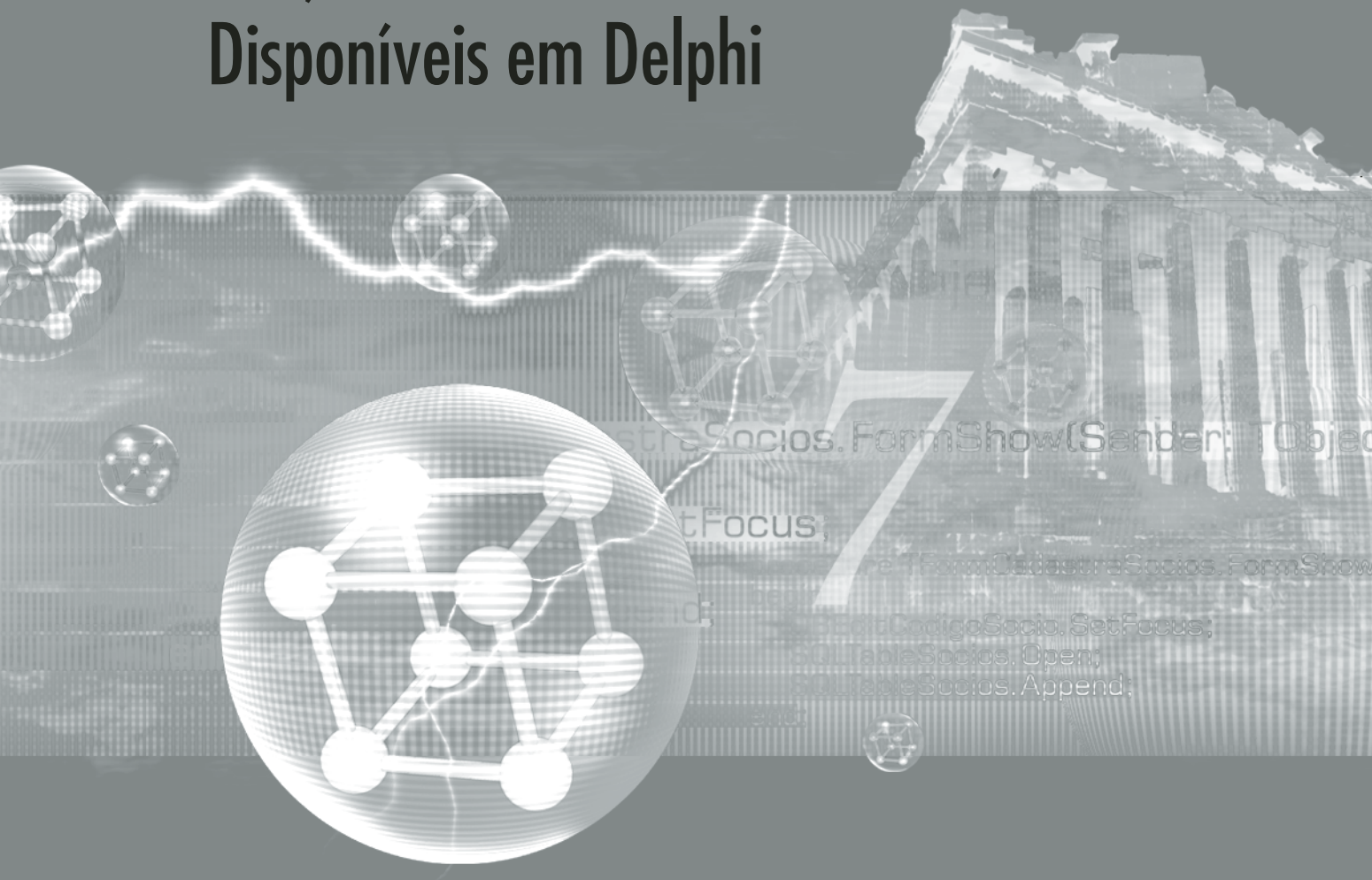
Referência

"Foram mantidas as referências da classe TQuickReport pelo fato da biblioteca ainda estar disponível para instalação no CD do Delphi 7".



Capítulo 39

Funções Matemáticas Disponíveis em Delphi



Neste capítulo, serão apresentadas as principais funções matemáticas disponíveis na linguagem Object Pascal. Destina-se principalmente àqueles que desejam migrar programas de outras linguagens para o Delphi.

ArcCos

Descrição

Essa função retorna o valor do ângulo (entre 0 e p radianos) cujo valor do cosseno (entre -1 e +1) é passado como parâmetro.

Declaração

```
function ArcCos(X: Extended): Extended;
```

ArcCosh

Descrição

Essa função retorna o valor real cujo cosseno hiperbólico (maior ou igual a +1) é passado como parâmetro.

Declaração

```
function ArcCosh(X: Extended): Extended;
```

ArcSin

Descrição

Essa função retorna o valor do ângulo (entre -p/2 e p/2 radianos) cujo valor do seno (entre -1 e +1) é passado como parâmetro.

Declaração

```
function ArcSin(X: Extended): Extended;
```

ArcSinh

Descrição

Essa função retorna o valor real cujo seno hiperbólico é passado como parâmetro.

Declaração

```
function ArcSinh(X: Extended): Extended;
```

ArcTanh

Descrição

Essa função retorna o valor real cuja tangente hiperbólica (entre -1 e +1) é passada como parâmetro.

Declaração

```
function ArcTanh(X: Extended): Extended;
```

ArcTan2

Descrição

Essa função retorna o ângulo cuja tangente (ArcTan(Y/X)) e quadrante são passados como parâmetros.

Declaração

```
function ArcTan2(Y, X: Extended): Extended;
```

Ceil

Descrição

Essa função retorna o menor inteiro maior ou igual ao valor passado como parâmetro.

Declaração

```
function Ceil(X: Extended):Integer;
```

Cosh

Descrição

Essa função retorna o cosseno hiperbólico do ângulo cujo valor é passado como parâmetro.

Declaração

```
function Cosh(X: Extended): Extended;
```

Cotan

Descrição

Essa função retorna a cotangente do ângulo cujo valor é passado como parâmetro.

Declaração

```
function Cotan(X: Extended): Extended;
```

CycleToRad

Descrição

Essa função converte em radianos o ângulo cujo valor em ciclos é passado como parâmetro.

Declaração

```
function CycleToRad(Cycles: Extended): Extended;
```

DegToRad

Descrição

Essa função converte em radianos o ângulo cujo valor em graus é passado como parâmetro.

Declaração

```
function DegToRad(Cycles: Extended): Extended;
```

Floor

Descrição

Essa função retorna o maior inteiro menor ou igual ao valor passado como parâmetro.

Declaração

```
function Floor(X: Extended):Integer;
```

Frexp

Descrição

Essa função retorna a mantissa e o expoente de um valor real passado como parâmetro.

Declaração

```
procedure Frexp(X: Extended; var Mantissa: Extended; var
Exponent: Integer) register;
```

GradToRad

Descrição

Essa função converte em radianos o ângulo cujo valor em grados é passado como parâmetro.

Declaração

```
function GradToRad(Cycles: Extended): Extended;
```

Hypot

Descrição

Essa função retorna o valor da hipotenusa de um triângulo, cujos valores dos catetos são passados como parâmetros.

Declaração

```
function Hypot(X, Y: Extended): Extended;
```

IntPower

Descrição

Essa função retorna uma potência inteira (passada como segundo parâmetro) de um valor-base (passado como primeiro parâmetro).

Declaração

```
function IntPower(Base: Extended; Exponent: Integer): Extended register;
```

Log2

Descrição

Essa função retorna o logaritmo na base 2 de um valor real passado como parâmetro.

Declaração

```
function Log2(X: Extended): Extended;
```

Log10

Descrição

Essa função retorna o logaritmo na base 10 de um valor real passado como parâmetro.

Declaração

```
function Log10(X: Extended): Extended;
```

LogN

Descrição

Essa função retorna o logaritmo numa base (cujo valor é passado como primeiro parâmetro) de um número real (cujo valor é passado como segundo parâmetro).

Declaração

```
function LogN(N, X: Extended): Extended;
```

Max

Descrição

Essa função retorna o maior de dois valores passados como parâmetro.

Declaração

Essa função é sobrecarregada, tendo as declarações apresentadas a seguir:

```
function Max(A,B: Integer): Integer; overload;
function Max(A,B: Int64): Int64; overload;
function Max(A,B: Single): Single; overload;
function Max(A,B: Double): Double; overload;
function Max(A,B: Extended): Extended; overload;
```

MaxIntValue

Descrição

Essa função retorna o maior valor dentre os armazenados em uma array de valores inteiros.

Declaração

```
function MaxIntValue(const Data: array of Integer): Integer;
```

MaxValue

Descrição

Essa função retorna o maior valor dentre os armazenados em uma array de valores reais.

Declaração

```
function MaxValue(const Data: array of Double): Double;
```

Mean

Descrição

Essa função retorna a média aritmética dos valores armazenados em uma array de valores reais.

Declaração

```
function Mean(const Data: array of Double): Extended;
```

MeanAndStdDev

Descrição

Essa função retorna a média aritmética e o desvio-padrão dos valores armazenados em uma array de valores reais.

Declaração

```
procedure MeanAndStdDev(const Data: array of Double; var Mean, StdDev: Extended);
```

Min

Descrição

Essa função retorna o menor de dois valores passados como parâmetro.

Declaração

Essa função é sobrecarregada, tendo as declarações apresentadas a seguir:

```
function Min(A,B: Integer): Integer; overload;
function Min(A,B: Int64): Int64; overload;
function Min(A,B: Single): Single; overload;
function Min(A,B: Double): Double; overload;
function Min(A,B: Extended): Extended; overload;
```

MinIntValue

Descrição

Essa função retorna o menor valor dentre os armazenados em uma array de valores inteiros.

Declaração

```
function MinIntValue(const Data: array of Integer): Integer;
```

MinValue

Descrição

Essa função retorna o menor valor dentre os armazenados em uma array de valores reais.

Declaração

```
function MinValue(const Data: array of Double): Double;
```

Norm

Descrição

Essa função retorna a norma euclidiana L-2 dentre os valores armazenados em uma array de valores reais.

Declaração

```
function Norm(const Data: array of Double): Extended;
```

Poly

Descrição

Essa função calcula o valor de um polinômio uniforme para o valor passado como primeiro parâmetro, sendo os coeficientes passados como segundo parâmetro na forma de uma array de valores reais.

Declaração

```
function Poly(X: Extended; const Coefficients: array of Double): Extended;
```

PopnStdDev

Descrição

Essa função calcula o valor do desvio-padrão de um conjunto de valores passados como parâmetro na forma de uma array de valores reais.

Declaração

```
function PopnStdDev(const Data: array of Double): Extended;
```

PopnVariance

Descrição

Essa função calcula o valor da variância de um conjunto de valores passados como parâmetro na forma de uma array de valores reais.

Declaração

```
function PopnVariance(const Data: array of Double): Extended;
```

Power

Descrição

Essa função retorna uma potência real (passada como segundo parâmetro) de um valor-base (passado como primeiro parâmetro).

Declaração

```
function Power(Base, Exponent: Extended): Extended;
```

RadToCycle

Descrição

Essa função converte em ciclos o ângulo cujo valor em radianos é passado como parâmetro.

Declaração

```
function RadToCycle(Radians: Extended): Extended;
```

RadToDeg

Descrição

Essa função converte em graus o ângulo cujo valor em radianos é passado como parâmetro.

Declaração

```
function RadToDeg(Radians: Extended): Extended;
```

RadToGrad

Descrição

Essa função converte em grados o ângulo cujo valor em radianos é passado como parâmetro.

Declaração

```
function RadToGrad(Radians: Extended): Extended;
```

RandG

Descrição

Essa função gera números aleatórios com distribuição gaussiana, sendo os valores da média e do desvio-padrão passados como parâmetros.

Declaração

```
function RandG(Mean, StdDev: Extended): Extended;
```

Sin

Descrição

Essa função calcula o seno de um ângulo, passado como parâmetro em radianos.

Declaração

```
function Sin(X: Extended): Extended;
```

SinCos

Descrição

Esse procedimento calcula o seno e o cosseno de um ângulo (passado em radianos, como primeiro parâmetro), sendo estes valores armazenados no segundo e terceiro parâmetros.

Declaração

```
procedure SinCos(Theta: Extended; var Sin, Cos: Extended); register;
```

Sinh

Descrição

Essa função retorna o seno hiperbólico do ângulo cujo valor é passado como parâmetro.

Declaração

```
function Sinh(X: Extended): Extended;
```

StdDev

Descrição

Essa função retorna o desvio-padrão dos valores armazenados em uma array de valores reais, passada como parâmetro.

Declaração

```
function StdDev(const Data: array of Double): Extended;
```

Sum

Descrição

Essa função retorna a soma dos valores armazenados em uma array de valores reais, passada como parâmetro.

Declaração

```
function Sum(const Data: array of Double): Extended;
```

SumInt

Descrição

Essa função retorna a soma dos valores armazenados em uma array de valores inteiros, passada como parâmetro.

Declaração

```
function SumInt(const Data: array of integer): Extended;
```

SumOfSquares

Descrição

Essa função retorna a soma dos quadrados dos valores armazenados em uma array de valores reais, passada como parâmetro.

Declaração

```
function SumOfSquares(const Data: array of Double): Extended;
```

SumsAndSquares

Descrição

Essa função retorna a soma dos valores e dos quadrados dos valores armazenados em uma array de valores reais, passada como parâmetro.

Declaração

```
procedure SumsAndSquares(const Data: array of Double; var Sum, SumOfSquares: Extended) register;
```

Tan

Descrição

Esse procedimento calcula a tangente de um ângulo em radianos.

Declaração

```
function Tan(X: Extended): Extended;
```

TanH

Descrição

Essa função calcula a tangente de um ângulo passado como parâmetro (em radianos).

Declaração

```
function TanH(X: Extended): Extended;
```

Variance

Descrição

Essa função retorna a variância dos valores armazenados em uma array de valores reais, passada como parâmetro.

Declaração

```
function Variance(const Data: array of Double): Extended;
```

Capítulo 40

Classes, Controles e Componentes

EAbort

Descrição

Essa classe permite a geração de exceções sem que seja exibida uma mensagem de erro em uma caixa de diálogo.

Unit

Na VCL e Na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EAbstractError

Descrição

Essa classe representa uma exceção gerada ao se tentar executar um método abstrato.

Unit

Na VCL e Na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp,
CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EAccessViolation

Descrição

Essa classe representa uma exceção gerada ao se tentar acessar um endereço de memória inválido.

Unit

Na VCL e Na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EArrayError

Descrição

Essa classe representa uma exceção gerada ao se tentar acessar um endereço inválido em uma array.

Unit

Na VCL:

```
mxArrays
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EAssertionFailed

Descrição

Essa classe representa uma exceção gerada ao se executar procedimentos de atribuição em expressões booleanas que retornam False.

Unit

Na VCL e na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EBitsError

Descrição

Essa classe representa uma exceção gerada ao se tentar realizar um acesso inválido a uma array de valores booleanos.

Unit

Na VCL e Na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EBrokerException

Descrição

Essa classe representa uma exceção gerada quando um objeto não consegue acessar um servidor.

Unit

Na VCL:

```
objbrkr
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

ECacheError

Descrição

Essa classe representa uma exceção gerada ao se tentar executar uma operação com o cache de um componente Decision Cube.

Unit

Na VCL:

```
mxstore
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EClassNotFound

Descrição

Essa classe representa uma exceção gerada ao se tentar carregar um componente de uma stream e este não está inserido na aplicação.

Unit

Na VCL e na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

ECommonCalendarError

Descrição

Essa classe representa uma exceção gerada ao se tentar atribuir um valor inválido a uma propriedade de um componente derivado da classe TCommonCalendar.

Unit

Na VCL:

```
ComCtrls
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EComponentError

Descrição

Essa classe representa uma exceção gerada ao se tentar registrar ou renomear um componente.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EControlC

Descrição

Essa classe representa uma exceção gerada ao se tentar encerrar uma aplicação do tipo console pressionando simultaneamente as teclas Ctrl e C.

Unit

Na VCL:

```
Sysutils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EConvertError

Descrição

Essa classe representa uma exceção gerada ao se tentar executar uma conversão de tipos inválida.

Unit

Na VCL e na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EDBClient

Descrição

Essa classe representa uma exceção gerada ao se utilizar um objeto da classe TclientDataSet (ou uma classe dela derivada por herança).

Unit

Na VCL e na CLX:

```
DBClient
```

Principais Propriedades

```
ErrorCode, HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EDBEditError

Descrição

Essa classe representa uma exceção gerada ao se tentar exibir, em um controle associado a um banco de dados, um valor incompat'vel com a máscara definida para o campo cujo valor é exibido pelo componente.

Unit

Na CLX:

```
QMask
```

Na VCL:

```
Mask
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EDBEngineError

Descrição

Essa classe representa uma exceção genérica, gerada ao se tentar executar uma função do BDE.

Unit

Na VCL:

```
DBtables
```

Principais Propriedades

```
ErrorCount, Errors, HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp e Destroy
```

Principais Eventos

Essa classe não possui eventos associados.

EDSWriter

Descrição

Essa classe representa uma exceção genérica, gerada ao se tentar manipular pacotes de dados por meio de um Dataset.

Unit

Na VCL e na CLX:

```
Provider
```

Principais Propriedades

```
ErrorCode, HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EDatabaseError

Descrição

Essa classe representa uma exceção genérica gerada ao se tentar acessar um banco de dados.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EDateTimeError

Descrição

Essa classe representa uma exceção genérica gerada ao se tentar atribuir valores inválidos a variáveis do tipo data/hora.

Unit

Na VCL e na CLX:

```
ComCtrls
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EDimIndexError

Descrição

Essa classe representa uma exceção genérica gerada ao se tentar utilizar um Índice inválido em uma dimensão de um componente da classe TDecisionSource.

Unit

Na VCL:

```
mxdb
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EDimensionMapError

Descrição

Essa classe representa uma exceção genérica gerada por incompatibilidade no formato dos dados armazenados em um dataset associado a um componente da classe TDecisionCube.

Unit

Na VCL:

```
mxdb
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EDivByZero

Descrição

Essa classe representa uma exceção gerada ao se tentar dividir um número inteiro por zero.

Unit

Na VCL e na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EExternal

Descrição

Essa classe representa uma exceção gerada externamente pelo sistema operacional.

Unit

Na VCL e na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EExternalException

Descrição

Essa classe representa uma exceção gerada, cujo código é inválido.

Unit

Na VCL e na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EFCreateError

Descrição

Essa classe representa uma exceção gerada ao se tentar criar um arquivo por meio de uma stream.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EFilerError

Descrição

Essa classe representa uma exceção genérica gerada ao se tentar manipular um arquivo por meio de uma stream.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EFOpenError

Descrição

Essa classe representa uma exceção gerada ao se tentar abrir um arquivo por meio de uma stream.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EHeapException

Descrição

Essa classe representa uma exceção gerada ao se tentar alocar memória no Heap do sistema operacional.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp,
CreateResHelp, FreeInstance
```

Principais Eventos

Essa classe não possui eventos associados.

EInOutError

Descrição

Essa classe representa uma exceção gerada ao se tentar efetuar operações de entrada e saída em arquivos, e desde que a diretiva de compilação $I+ esteja ativada. Valores na faixa 0-99 representam erros nativos do sistema operacional, ao passo que valores na faixa 100-149 representam erros definidos pela CLX.

Exemplos de código de erros da CLX:

Código do Erro	Significado
100	Fim de arquivo
101	Disco cheio
102	Variável não atribuída a arquivo
103	File not open
104	Arquivo não foi aberto para leitura
105	Arquivo não foi aberto para escrita
106	Erro na formatação de saída
107	Arquivo já aberto

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp,
CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

wEInterpreterError

Descrição

Essa classe representa uma exceção genérica gerada ao se manipular objetos COM.

Unit

Na VCL:

```
SConnect
```

Principais Propriedades
HelpContext e Message

Principais Métodos
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp

Principais Eventos

Essa classe não possui eventos associados.

EIntError

Descrição

Essa classe representa uma exceção genérica gerada ao se manipular valores inteiros.

Unit

Na VCL e na CLX:

SysUtils

Principais Propriedades
HelpContext e Message

Principais Métodos
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp,

Principais Eventos

Essa classe não possui eventos associados.

EIntfCastError

Descrição

Essa classe representa uma exceção gerada ao se tentar converter explicitamente uma interface, usando o operador "as".

Unit

Na VCL e na CLX:

SysUtils

Principais Propriedades
HelpContext e Message

Principais Métodos
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp

Principais Eventos

Essa classe não possui eventos associados.

EIntOverFlow

Descrição

Essa classe representa uma exceção gerada ao se tentar atribuir a uma variável inteira um valor cuja magnitude é superior àquela capaz de ser armazenada pelo tipo da variável.

Unit

Na VCL e na CLX:

SysUtils

Principais Propriedades
HelpContext e Message

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EInvalidArgument

Descrição

Essa classe representa uma exceção gerada ao se tentar efetuar uma operação inválida passando-se valores fora da faixa especificada em parâmetros de funções matemáticas.

Unit

Na VCL e na CLX:

```
Math
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EInvalidCast

Descrição

Essa classe representa uma exceção gerada ao se tentar converter explicitamente o tipo de uma variável ou objeto.

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EInvalidGraphic

Descrição

Essa classe representa uma exceção gerada ao se tentar manipular um arquivo gráfico cujo formato é inválido.

Unit

Na VCL:

```
Graphics
```

Na CLX:

```
QGraphics
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EInvalidGraphicOperation

Descrição

Essa classe representa uma exceção gerada ao se tentar manipular um gráfico.

Unit

Na VCL:

```
Graphics
```

Na CLX:

```
QGraphics
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EInvalidGridOperation

Descrição

Essa classe representa uma exceção gerada ao se tentar manipular um componente do tipo StringGrid ou componentes derivados por herança.

Unit

Na VCL:

```
Grids
```

Na CLX:

```
QGrids
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EInvalidImage

Descrição

Essa classe representa uma exceção gerada ao se tentar manipular um arquivo de recurso que armazena uma imagem.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EInvalidOp

Descrição

Essa classe representa uma exceção gerada ao se tentar efetuar uma operação inválida com números de ponto flutuante.

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EInvalidOperation

Descrição

Essa classe representa uma exceção gerada ao se tentar efetuar uma operação inválida com um componente.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EInvalidPointer

Descrição

Essa classe representa uma exceção gerada ao se tentar efetuar uma operação inválida com um ponteiro.

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

HelpContext e Message

Principais Métodos

Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp

Principais Eventos

Essa classe não possui eventos associados.

EListError

Descrição

Essa classe representa uma exceção gerada ao se tentar efetuar uma operação com listas de objetos, strings ou listas de strings.

Unit

Na VCL e na CLX:

Classes

Principais Propriedades

HelpContext e Message

Principais Métodos

Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp

Principais Eventos

Essa classe não possui eventos associados.

ELowCapacityError

Descrição

Essa classe representa uma exceção gerada ao se tentar alocar mais memória do que a existente para um componente TDecisionCube.

Unit

Na VCL:

mxArrays

Principais Propriedades

HelpContext e Message

Principais Métodos

Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp

Principais Eventos

Essa classe não possui eventos associados.

EMathError

Descrição

Essa classe representa uma exceção genérica gerada ao se tentar efetuar uma operação inválida com números de ponto flutuante.

Unit

Na VCL e na CLX:

SysUtils

Principais Propriedades

HelpContext e Message

Principais Métodos

`Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp`

Principais Eventos

Essa classe não possui eventos associados.

EMCIDeviceError

Descrição

Essa classe representa uma exceção gerada ao se tentar acessar um driver MCI (Media Control Interface).

Unit

Na VCL:

`mplayer`

Principais Propriedades

`HelpContext e Message`

Principais Métodos

`Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp`

Principais Eventos

Essa classe não possui eventos associados.

EMenuError

Descrição

Essa classe representa uma exceção gerada ao se manipular itens de menu.

Unit

Na VCL:

`Menus`

Na CLX:

`QMenus`

Principais Propriedades

`HelpContext e Message`

Principais Métodos

`Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp`

Principais Eventos

Essa classe não possui eventos associados.

EMonthCalError

Descrição

Essa classe representa uma exceção gerada ao se tentar atribuir valores a um componente

`TMonthCalendar.`

Unit

Na VCL:

`ComCtrls`

Principais Propriedades

`HelpContext e Message`

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

ENoResultSet

Descrição

Essa classe representa uma exceção gerada ao se tentar acessar um banco de dados por meio de uma declaração SQL em um componente TQuery.

Unit

Na VCL:

```
DBtables
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EOLECtrlError

Descrição

Essa classe representa uma exceção gerada ao se tentar utilizar um controle ActiveX em uma aplicação.

Unit

Na VCL:

```
Olectrls
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EOLEError

Descrição

Essa classe representa uma exceção gerada ao se efetuarem operações de baixo nível com controles OLE.

Unit

Na VCL:

```
ComObj
```

Principais Propriedades

```
HelpContext e Message
```

Principais Métodos

```
Create, CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EOLEException

Descrição

Essa classe representa uma exceção gerada ao se tentar acessar propriedades e métodos em objetos de automação OLE.

Unit

Na VCL:

```
ComObj
```

Principais Propriedades

```
ErrorCode, HelpContext, HelpKeyword, HelpTypeHelpFile, Message e Source
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EOLESysError

Descrição

Essa classe representa uma exceção gerada ao se tentar acessar interfaces do tipo IDispatch.

Unit

Na VCL:

```
ComObj
```

Principais Propriedades

```
ErrorCode, HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EOSError

Descrição

Essa classe representa uma exceção genérica gerada pelo sistema operacional (Windows ou Linux).

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EOutOfMemory

Descrição

Essa classe representa uma exceção gerada ao se tentar alocar memória.

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EOutOfResources

Descrição

Essa classe representa uma exceção gerada ao se tentar alocar um handle no sistema operacional.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EOutlineError

Descrição

Essa classe representa uma exceção gerada ao se manipular componentes TOutLine.

Unit

Na VCL:

```
Outline
```

Principais Propriedades

```
HelpContexteMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EOverflow

Descrição

Essa classe representa uma exceção gerada ao se tentar atribuir valores cuja magnitude é superior à suportada pelo tipo de uma variável de ponto flutuante.

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EPackageError

Descrição

Essa classe representa uma exceção gerada ao se tentar manipular pacotes de código.

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EParseError

Descrição

Essa classe representa uma exceção gerada ao se tentar converter valores no formato texto para binário e vice-versa.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EPrinter

Descrição

Essa classe representa uma exceção gerada ao se realizar um trabalho de impressão.

Unit

Na VCL:

```
Printers
```

Na CLX:

```
QPrinters
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EPrivilege

Descrição

Essa classe representa uma exceção gerada ao se tentar executar uma instrução inválida para o processador.

Unit

Na VCL e na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EPropertyError

Descrição

Essa classe representa uma exceção gerada ao se tentar atribuir um valor a uma propriedade.

Unit

Na VCL e na CLX:

```
TypInfo
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EPropReadOnly

Descrição

Essa classe representa uma exceção gerada ao se tentar atribuir um valor a uma propriedade que use automação OLE quando a propriedade é apenas de escrita.

Unit

Na VCL:

```
Sysutils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EPropWriteOnly

Descrição

Essa classe representa uma exceção gerada ao se tentar ler o valor de uma propriedade que use automação OLE quando a propriedade é apenas de leitura.

Unit

Na VCL:

```
Sysutils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

ERangeError

Descrição

Essa classe representa uma exceção gerada ao se obterem valores inteiros fora da faixa permitida para o seu tipo, principalmente como resultado de operações aritméticas.

Unit

Na VCL e na CLX:

```
Sysutils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EReadError

Descrição

Essa classe representa uma exceção gerada ao se tentar ler valores de uma stream.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EReconcileError

Descrição

Essa classe representa uma exceção gerada ao se manipularem componentes do tipo TClientDataSet ou dele derivados por herança, ao se tentar efetuar uma operação de atualização de dados.

Unit

Na VCL e na CLX:

```
DBclient
```

Principais Propriedades

```
Context, ErrorCode, HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

ERegistryException

Descrição

Essa classe representa uma exceção gerada ao se manipular o registro do Windows.

Unit

Na VCL:

```
Registry
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EResNotFound

Descrição

Essa classe representa uma exceção gerada ao se tentar acessar um recurso armazenado em um arquivo DFM (VCL), XFM (CLX) ou RES.

Unit

Na VCL e na CLX:

```
classes
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

ESocketConnectionError

Descrição

Essa classe representa uma exceção gerada ao se tentar enviar ou receber mensagens usando um componente TSocketConnection.

Unit

Na VCL:

```
Sconnect
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

ESocketError

Descrição

Essa classe representa uma exceção gerada ao se manipularem objetos de classes derivadas de TCustomWinSocket.

Unit

Na VCL:

```
Sockets
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EStackOverflow

Descrição

Essa classe representa uma exceção gerada quando ocorre um estouro da pilha de memória do sistema operacional.

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EStreamError

Descrição

Essa classe representa uma exceção gerada quando se manipulam streams.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EStringListError

Descrição

Essa classe representa uma exceção gerada quando se tenta acessar um item de um componente Listbox com um Índice inválido.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EThread

Descrição

Essa classe representa uma exceção gerada quando se manipulam múltiplas threads em uma aplicação. Não é mais empregada, sendo sua definição mantida para manter a compatibilidade com código escrito em versões anteriores do Delphi.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

ETreeViewError

Descrição

Essa classe representa uma exceção gerada quando se tenta acessar um elemento (nó) de um componente TTreeView.

Unit

Na VCL:

```
Comctrls
```

Na CLX:

```
QComctrls
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EUnderflow

Descrição

Essa classe representa uma exceção gerada ao se tentar atribuir valores cuja magnitude é inferior à suportada pelo tipo de uma variável de ponto flutuante.

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EUnsupportedTypeError

Descrição

Essa classe representa uma exceção gerada ao se tentar acrescentar uma dimensão ao cache de um componente TDecisionCube.

Unit

Na VCL:

```
mxArrays
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EUpdateError

Descrição

Essa classe representa uma exceção gerada ao se tentar manipular um componente

```
Tprovider, atualizando os dados por ele manipulados.
```

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Context, ErrorCode, OriginalException, HelpContext, HelpKeyword, HelpTypeMessage, PreviousError
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EVariantError

Descrição

Essa classe representa uma exceção gerada ao se manipularem dados do tipo Variant.

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EWin32Error

Descrição

Essa classe representa uma exceção genérica gerada pelo Windows. Atualmente substituída pela classe EOSError, que representa uma exceção genérica gerada pelo sistema operacional (Windows ou Linux).

Unit

Na VCL:

```
SysUtils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EWriteError

Descrição

Essa classe representa uma exceção gerada ao se tentar escrever um valor em uma stream.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

EZeroDivide

Descrição

Essa classe representa uma exceção gerada ao se tentar dividir um número de ponto flutuante por zero.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

Exception

Descrição

Essa classe representa a classe-base para todas as classes usadas para representar exceções.

Unit

Na VCL e na CLX:

```
SysUtils
```

Principais Propriedades

```
HelpContext, HelpKeyword, HelpTypeMessage;
```

Principais Métodos

```
CreateFmt, CreateFmtHelp, CreateHelp, CreateRes, CreateResFmt, CreateResFmtHelp, CreateResHelp
```

Principais Eventos

Essa classe não possui eventos associados.

TAction

Descrição

Essa classe representa uma ação definida em um objeto da classe TActionList.

Unit

Na VCL:

```
ActnList
```

Na CLX:

```
QActnList
```

Principais Propriedades

```
Autocheck, Caption, Checked, DisableIfNoHandler, Enabled, GroupIndex, HelpContext, HelpKeyword,
HelpTypeHelpKeyword, HelpType, Hint, ImageIndex, Name, ShortCut, Visible, Category e Index
```

Principais Métodos

```
Create, Destroy, DoHint, Execute, GetParentComponent, Update, ExecuteTarget, HandlesTarget,
RegisterChanges, UnRegisterChanges, UpdateTarget
```

Principais Eventos

```
OnHint, OnExecute e OnUpdate
```

TActionList

Descrição

Essa classe representa um conjunto de ações (cada ação é representada por um objeto da classe TAction) compartilhadas por controles, componentes e itens de menu.

Unit

Na VCL:

```
ActnList
```

Na CLX:

```
QActnList
```

Principais Propriedades

```
ActionCount, Actions, Images
```

Principais Métodos

```
Create, Destroy, ExecuteAction, IsShortCut e UpdateAction
```

Principais Eventos

OnHint, OnExecute e OnUpdate

TADOCommand

Descrição

Este componente permite executar comandos sql e stored procedures em tabelas de bancos de dados através do Mecanismo Activex Data Objects (ADO).

Unit

Na VCL:

AdoDB

Principais Propriedades

CommandObject, CommandText, CommandTimeout, CommandType, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Connection, ConnectionString, DesignInfo, ExecuteOptions, Name, Owner, ParamCheck, Parameters, Prepared, Properties, States, Tag, VCLComObject

Principais Métodos

AfterConstruction, Assign, Assign, BeforeDestruction, Cancel, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, Create, DefaultHandler, Destroy, DestroyComponents, Destroying, Dispatch, Execute, ExecuteAction, FieldAddress, FindComponent, Free, FreeInstance, FreeNotification, FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName, NewInstance, RemoveComponent, SafeCallException, UpdateAction

Principais Eventos

Este componente não possui procedimentos associados a eventos

TADOConnection

Descrição

Este componente É responsável pelo acesso a bancos de dados através do Mecanismo Activex Data Objects (ADO).

Unit

Na VCL:

AdODB

Principais Propriedades

Attributes, CommandCount, Commands, CommandTimeout, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Connected, ConnectionObject, ConnectionString, ConnectionTimeout, ConnectOptions, CursorLocation, DataSetCount, DataSets, DefaultDatabase, DesignInfo, Errors, InTransaction, IsolationLevel, LoginPrompt, Mode, Owner, Properties, Provider, State, Tag, VCLComObject, Version

Principais Métodos

AfterConstruction, Assign, BeforeDestruction, BeginTrans, Cancel, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, Close, CloseDataSets, CommitTrans, Create, DefaultHandler, Destroy, DestroyComponents, Destroying, Dispatch, Execute, ExecuteAction, FieldAddressm FindComponent, Free, FreeInstance, FreeNotification, FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GetProcedureNames, GetTableNames, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName, NewInstance, Open, OpenSchema, RemoveComponent, RollbackTrans, SafeCallException, UpdateAction

Principais Eventos

AfterConnect, AfterDisconnect, BeforeConnect, BeforeDisconnect, OnBeginTransComplete, OnCommitTransComplete, OnConnectComplete, OnDisconnect, OnExecuteComplete, OnInfoMessage, OnLogin, OnRollbackTransComplete, OnWillConnect, OnWillExecute

TADODataSet

Descrição

Este componente permite acesso direto ou via declarações sql a tabelas de bancos de dados através do Mecanismo Activex Data Objects (ADO).

Unit

Na VCL:

AdoDB

Principais Propriedades

Active, ActiveRecord, AggFields, AutoCalcFields, BlobFieldCount, BlockReadSize, Bof, Bookmark, BookmarkSize, BufferCount, Buffers, CacheSize, CalcBuffer, CalcFieldsSize, CanModify, CommandText, CommandTimeout, CommandType, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Connection, ConnectionString, Constraints, CurrentRecord, CursorLocation, CursorType, DataSetField, DefaultFields, Designer, DesignerData, DesignInfo, Eof, ExecuteOptions, FieldCount, FieldDefList, FieldDefs, FieldList, FieldNoOfs, Fields, FieldValues, Filter, Filtered, FilterGroup, FilterOptions, Found, IndexFieldCount, IndexFields, InternalCalcFields, LockType, MarshalOptions, MaxRecords, Modified, Name, NestedDataSetClass, NestedDataSets, ObjectView, Owner, ParamCheck, Parameters, Prepared, Properties, RDSConnection, RecNo, RecordCount, RecordSet, RecordSetState, RecordSize, RecordStatus, Reserved, Sort, SparseArrays, State, Tag, VCLComObject

Principais Métodos

ActiveBuffer, AfterConstruction, Append, AppendRecord, Assign, BeforeDestruction, BookmarkValid, Cancel, CancelBatch, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Clone, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CreateDataset, CursorPosChanged, DefaultHandler, Delete, DeleteRecords, Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EnableControls, ExecuteAction, FieldAddress, FieldByName, FilterOnBookmarks, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetIndexNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GotoBookmark, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsLinkedTo, IsSequenced. Last, LoadFromFile, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, NextRecordset, Open, Post, Prior, Refresh, RemoveComponent, Resync, SafeCallException, SaveToFile, Seek, SetFields, Supports, Translate, UpdateAction, UpdateBatch, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnEndOfRecordset, OnFetchComplete, OnFetchProgress, OnFieldChangeComplete, OnFilterRecord, OnMoveComplete, OnNewRecord, OnPostError, OnRecordChangeComplete, OnRecordsetChangeComplete, OnWillChangeField, OnWillChangeRecord, OnWillChangeRecordset, OnWillMove

TADOQuery

Descrição

Este componente permite acesso via declarações sql a tabelas de bancos de dados através do Mecanismo Activex Data Objects (ADO).

Unit

Na VCL:

AdoDB

Principais Propriedades

Active, ActiveRecord, AggFields, AutoCalcFields, BlobFieldCount, BlockReadSize, Bof, Bookmark, BookmarkSize, BufferCount, Buffers, CacheSize, CalcBuffer, CalcFieldsSize, CanModify, CommandText, CommandTimeout, CommandType, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Connection, ConnectionString, Constraints, CurrentRecord, CursorLocation,

CursorType, DataSetField, DataSource, DefaultFields, Designer, DesignerData, DesignInfo, Eof, ExecuteOptions, FieldCount, FieldDefList, FieldDefs, FieldList, FieldNoOfs, Fields, FieldValues, Filter, Filtered, FilterGroup, FilterOptions, Found, IndexFieldCount, IndexFields, InternalCalcFields, LockType, MarshalOptions, MaxRecords, Modified, Name, NestedDataSetClass, NestedDataSets, ObjectView, Owner, ParamCheck, Parameters, Prepared, Properties, RecNo, RecordCount, RecordSet, RecordSetState, RecordSize, RecordStatus, Reserved, RowsAffected, Sort, SparseArrays, SQL, State, Tag, VCLComObject

Principais Métodos

ActiveBuffer, AfterConstruction, Append, AppendRecord, Assign, BeforeDestruction, BookmarkValid, Cancel, CancelBatch, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Clone, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CursorPosChanged, DefaultHandler, Delete, DeleteRecords, Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EnableControls, ExecSQL, ExecuteAction, FieldAddress, FieldByName, FilterOnBookmarks, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GotoBookmark, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsLinkedTo, IsSequenced. Last, LoadFromFile, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, NextRecordset, Open, Post, Prior, Refresh, RemoveComponent, Resync, SafeCallException, SaveToFile, Seek, SetFields, Supports, Translate, UpdateAction, UpdateBatch, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnEndOfRecordset, OnFetchComplete, OnFetchProgress, OnFieldChangeComplete, OnFilterRecord, OnMoveComplete, OnNewRecord, OnPostError, OnRecordChangeComplete, OnRecordsetChangeComplete, OnWillChangeField, OnWillChangeRecord, OnWillChangeRecordset, OnWillMove

TADOStoredProc

Descrição

Esse componente permite que uma aplicação desenvolvida em Delphi execute procedimentos armazenados em servidores, através do mecanismo Activex Data Objects (ADO).

Unit

Na VCL:

ADODB

Principais Propriedades

Active, ActiveRecord, AggFields, AutoCalcFields, BlobFieldCount, BlockReadSize, Bof, Bookmark, BookmarkSize, BufferCount, Buffers, CacheSize, CalcBuffer, CalcFieldsSize, CanModify, CommandText, CommandTimeout, CommandType, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Connection, ConnectionString, Constraints, CurrentRecord, CursorLocation, CursorType, DataSetField, DataSource, DefaultFields, Designer, DesignerData, DesignInfo, Eof, ExecuteOptions, FieldCount, FieldDefList, FieldDefs, FieldList, FieldNoOfs, Fields, FieldValues, Filter, Filtered, FilterGroup, FilterOptions, Found, IndexFieldCount, IndexFields, InternalCalcFields, LockType, MarshalOptions, MaxRecords, Modified, Name, NestedDataSetClass, NestedDataSets, ObjectView, Owner, ParamCheck, Parameters, Prepared, ProcedureName, Properties, RecNo, RecordCount, RecordSet, RecordSetState, RecordSize, RecordStatus, Reserved, Sort, SparseArrays, State, Tag, VCLComObject

Principais Métodos

ActiveBuffer, AfterConstruction, Append, AppendRecord, Assign, BeforeDestruction, BookmarkValid, Cancel, CancelBatch, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Clone, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CursorPosChanged, DefaultHandler, Delete, DeleteRecords, Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EnableControls, ExecuteAction, FieldAddress, FieldByName, FilterOnBookmarks, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields,

GetFieldData, GetFieldList, GetFieldNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GotoBookmark, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsLinkedTo, IsSequenced. Last, LoadFromFile, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, NextRecordset, Open, Post, Prior, Refresh, RemoveComponent, Resync, SafeCallException, SaveToFile, Seek, SetFields, Supports, Translate, UpdateAction, UpdateBatch, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnEndOfRecordset, OnFetchComplete, OnFetchProgress, OnFieldChangeComplete, OnFilterRecord, OnMoveComplete, OnNewRecord, OnPostError, OnRecordChangeComplete, OnRecordsetChangeComplete, OnWillChangeField, OnWillChangeRecord, OnWillChangeRecordset, OnWillMove

TADOTABLE

Descrição

Este componente permite acesso direto a tabelas de bancos de dados através do Mecanismo Activex Data Objects (ADO).

Unit

Na VCL:

AdoDB

Principais Propriedades

Active, ActiveRecord, AggFields, AutoCalcFields, BlobFieldCount, BlockReadSize, Bof, Bookmark, BookmarkSize, BufferCount, Buffers, CacheSize, CalcBuffer, CalcFieldsSize, CanModify, CommandText, CommandTimeout, CommandType, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Connection, ConnectionString, Constraints, CurrentRecord, CursorLocation, CursorType, DataSetField, DataSource, DefaultFields, Designer, DesignerData, DesignInfo, Eof, ExecuteOptions, FieldCount, FieldDefList, FieldDefs, FieldList, FieldNoOfs, Fields, FieldValues, Filter, Filtered, FilterGroup, FilterOptions, Found, IndexFieldCount, IndexFieldNames, IndexFields, InternalCalcFields, LockType, MarshalOptions, MasterFields, MasterSource, MaxRecords, Modified, Name, NestedDataSetClass, NestedDataSets, ObjectView, Owner, ParamCheck, Parameters, Prepared, Properties, ReadOnly, RecNo, RecordCount, RecordSet, RecordSetState, RecordSize, RecordStatus, Reserved, Sort, SparseArrays, State, TableDirect, TableName, Tag, VCLComObject

Principais Métodos

ActiveBuffer, AfterConstruction, Append, AppendRecord, Assign, BeforeDestruction, BookmarkValid, Cancel, CancelBatch, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Clone, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CursorPosChanged, DefaultHandler, Delete, DeleteRecords, Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EnableControls, ExecuteAction, FieldAddress, FieldByName, FilterOnBookmarks, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetIndexNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GotoBookmark, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsLinkedTo, IsSequenced. Last, LoadFromFile, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, NextRecordset, Open, Post, Prior, Refresh, RemoveComponent, Resync, SafeCallException, SaveToFile, Seek, SetFields, Supports, Translate, UpdateAction, UpdateBatch, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnEndOfRecordset, OnFetchComplete, OnFetchProgress, OnFieldChangeComplete, OnFilterRecord, OnMoveComplete, OnNewRecord, OnPostError, OnRecordChangeComplete, OnRecordsetChangeComplete, OnWillChangeField, OnWillChangeRecord, OnWillChangeRecordset, OnWillMove

TAnimate

Descrição

Esse componente permite que se executem animações armazenadas em arquivos AVI (sem execução de som), componentes e itens de menu.

Unit

Na VCL:

```
ComCtrls
```

Principais Propriedades

```
Active, Cente, CommonAVI, FileName, FrameCount, FrameHeight, FrameWidth, Open, Repetitions,
ResHandle, ResID, ResName, StartFrame, StopFrame, Timers, Transparent
```

Principais Métodos

```
CanAutoSize, Create, Play, Reset, Seek, Stop
```

Principais Eventos

```
OnClose, OnOpen, OnStart e OnStop
```

TApplication

Descrição

Esse componente está presente em todas as aplicações do Delphi. É automaticamente incorporado a um projeto durante a sua criação.

Unit

Na VCL:

```
Forms
```

Na CLX:

```
QForms
```

Principais Propriedades

```
Active, ComponentCount, ComponentIndex, Components, ExeName, Handle, HelpFile, Hint, HintColr,
HintPause, Icon, MainForm, Name, Owner, ShowHint, Tag, Terminated, Title
```

Principais Métodos

```
Create, CreateForm, Destroy, FindComponent, Free, HandleException, HelpCommand, HelpContext,
HelpKeyword, HelpTypeHelpJunt, InsertComponent, MessageBox, Minimize, NormalizeTopMosts,
ProcessMessages, RemoveComponent, Restore, RestoreTopMosts, Run, ShowException e Terminate
```

Principais Eventos

```
OnActivate, OnDeactivate, OnException, OnHelp, OnHint, OnIdle e OnMessage
```

TApplicationEvents

Descrição

Este componente intercepta eventos da aplicação.

Unit

Na VCL:

```
AppEvents
```

Principais Propriedades

```
ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DesignInfo,
Name, Owner, Tag e VCLComObject
```

Principais Métodos

```
Activate, AfterConstruction, Assign, BeforeDestruction, CancelDispatch, ClassInfo, ClassName,
ClassNameIs, ClassParent, ClassType, CleanupInstance, Create, DefaultHandler, Destroy,
DestroyComponents, Destroying, Dispatch, FieldAddress, FindComponent, Free, FreeInstance,
FreeNotification, FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath,
GetParentComponent, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize,
MethodAddress, MethodName, NewInstance, RemoveComponent, SafeCallException
```

Principais Eventos

```
OnActionExecute, OnActionUpdate, OnActivate, OnDeactivate, OnException, OnHelp, OnHint, OnIdle,
OnMessage, OnMinimize, OnRestore, OnShortCut, OnShowHint
```

TAUTOINCFIELD

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um valor inteiro auto-incrementado, podendo assumir valores entre -2147483648 e 2147483647.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Alignment, AsBoolean, AsDateTime, AsFloat, AsInteger, AsString, AsSQLTimeStamp, Calculated,
CanModify, DataSet, DataSize, DataType, DisplayFormat, DisplayLabel, DisplayName, DisplayText,
DisplayWidth, EditFormat, FieldName, FieldNo, Index, IsIndexField, IsNull, MaxValue. MinValue, Name,
Owner, ReadOnly, Required, Size, Tag, Text, Value e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, Create, Destroy, FocusControl, Free, GetData, IsValidChar e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate
```

TBASEREPORT

```
Unit
```

Na VCL:

```
RpBase
```

Na CLX:

```
QRpBase
```

Descrição

Esta classe é a classe-base para os principais componentes de visualização do Rave Reports.

Principais Propriedades

```
Aborted, AccuracyMethod, AscentHeight, Bins, BKColor, Bold, BottomWaste, BoxLineColor, Canvas,
Collate, ColumnEnd, ColumnLinesLeft, ColumnNum, Columns, ColumnStart, ColumnWidth, Copies,
CurrentPage, CurrentPass, CursorXPos, CursorYPos, DescentHeight, DeviceName, DevMode, DriverName,
Duplex, FileName, FirstPage, FontAlign, FontBaseline, FontBottom, FontCharset, FontColor,
FontHandle, FontHeight, FontName, FontPitch, FontRotation, Fonts, FontSize, FontTop, FontWidth,
FrameMode, GridVert, Italic, LastPage, LeftWaste, LineBottom, LineHeight, LineHeightMethod,
LineMiddle, LineNum, LinesPerInch, LineTop, MacroData, MarginBottom, MarginLeft, MarginRight,
MarginTop, MaxCopies, NoBufferLine, NoNTColorFix, NoPrinterPageHeight, NoPrinterPageWidth,
Orientation, OriginX, OriginY, OutputInvalid, OutputName, PageHeight, PageInvalid, PageWidth,
Papers, PIVar, Port, PrinterIndex, Printers, Printing, ReportDateTime, RightWaste, ScaleX, ScaleY,
SectionBottom, SectionLeft, SectionRight, SectionTop, Selection, ShadowDepth, StatusFormat,
StatusLabel, StatusText, Stream, StreamMode, Strikeout, Subscript, Superscript, TabColor,
TabJustify, TabShade, TextBKMode, Title, TopWaste, TotalPasses, TransparentBitmaps, TruncateText,
Underline, Units, UnitsFactor, Version, XDPI, XPos, YDPI, YPos
```

Principais Métodos

Abort, AbortPage, AdjustLine, AllowAll, AllowPreviewOnly, AllowPrinterOnly, Arc, AssignFont, BrushCopy, CalcGraphicHeight, CalcGraphicWidth, Chord, ClearAllTabs, ClearColumns, ClearTabs, CopyRect, CR, Create, CreateBrush, CreateFont, CreatePen, CreatePoint, CreateRect, Destroy, DrawFocusRect, Draw, Ellipse, Execute, FillRect, Finish, FinishTabBox, FloodFill, FrameRect, GetMemoLine, GetNextLine, GetTab, GotoFooter, GotoHeader, GotoXY, GraphicFieldToBitmap, Home, LF, LinesLeft, LineTo, Macro, MemoLines, MoveTo, NewColumn, NewLine, NewPage, NoPrinters, Pie, Polygon, Polyline, PopFont, PopPos, PopTabs, Print, PrintBitmap, PrintBitmapRect, PrintBlock, PrintCenter, PrintCharJustify, PrintData, PrintDataStream, PrintFooter, PrintHeader, PrintImageRect, PrintJustify, PrintLeft, PrintLn, PrintMemo, PrintRight, PrintTab, PrintXY, PushFont, PushPos, PushTabs, RecoverPrinter, Rectangle, RegisterGraphic, ReleasePrinter, Reset, ResetLineHeight, ResetPrinter, ResetSection, ResetTabs, RestoreFont, RestorePos, RestoreTabs, ReuseGraphic, RoundRect, SaveFont, SavePos, SaveTabs, SelectBin, SelectPaperO SelectPrinter, SetBrush, SetColumns, SetColumnWidth, SetFont, SetPaperSize, SetPen, SetPIVar, SetTab, SetTopOfPage, ShadeToColor, ShowPrintDialog, ShowPrinterSetupDialog, Start, StretchDraw, SupportBin, SupportCollate, SupportDuplex, SupportOrientation, SupportPaper, Tab, TabEnd, TabStart, TabWidth, TextRect, TextWidth, UnregisterGraphic, UpdateStatus, XD2U, XI2D, XI2U, XU2D, XU2I, YD2I, YD2U, YI2D, YI2U, YU2D, YU2I

Principais Eventos

OnAfterPrint, OnBeforePrint, OnDecodeImage, OnNewColumn, OnNewPage, OnPrint, OnPrintFooter, OnPrintHeader, OnPrintPage

TBatchMove

Descrição

Esse componente permite que se realizem operações sobre grupos de registros ou tabelas inteiras.

Unit

Na VCL:

DBTables

Principais Propriedades

AbortOnKeyViol, AbortOnProblem, ChangedCount, ChangedTableName, Destination, KeyViolCount, KeyViolTableName, Mappings, Mode, MovedCount, Name, Owner, ProblemCount, ProblemTableName, RecordCount, Source, Translitarate e Tag

Principais Métodos

Execute

Principais Eventos

Esse componente não possui eventos associados.

TBCDField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um valor em código decimal binário, com uma precisão de 18 algarismos significativos.

Unit

Na VCL e na CLX:

DB

Principais Propriedades

Alignment, AsBoolean, AsDateTime, AsFloat, AsInteger, AsString, Calculated, CanModify, Currency, DataSet, DataSize, DataType, DisplayFormat, DisplayLabel, DisplayName, DisplayText, DisplayWidth, EditFormat, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IndexField, IsNull, MaxValue. MinValue, Name, Owner, Precision, ReadOnly, Required, Size, Tag, Text, Value e Visible

Principais Métodos

Assign, AssignValue, Clear, Create, Destroy, FocusControl, Free, GetData, IsValidChar e SetData

Principais Eventos

OnChange, OnGetText, OnSetText e OnValidate

TBEVEL

Descrição

O controle TBevel permite que sejam colocados quadros, caixas e linhas chanfrados (com aspecto tridimensional) nos formulários que compõem a sua aplicação.

Unit

Na VCL:

ExtCtrls

Na CLX:

QExtCtrls

Principais Propriedades

Align, BoundsRect, ComponentIndex, Components, ControlCount, Controls, Handle, Height, HelpContext, HelpKeyword, HelpTypeHelpKeyword, HelpType, Hint, Left, Name, Owner, Parent, ParentShowHint, Shape, ShowHint, Style, Tag, Top, Visible, Width

Principais Métodos

BeginDrag, BringToFront, ClientToScreen, Dragging, EndDrag, Hide, Refresh, Repaint, ScreenToClient, SendToBack, SetBounds, Show e Update

TBITBTN

Descrição

O controle TBitBtn é um botão de pressionamento que permite a inclusão de um bitmap na sua face.

Unit

Na VCL:

Buttons

Na CLX:

QButtons

Principais Propriedades

Align, BoundsRect, Cancel, Caption, ComponentIndex, Cursor, Default, DragCursor, DragMode, Enabled, Font, Glyph, Height, HelpContext, HelpKeyword, HelpTypeHint, Kind, Layout, Left, Margin, ModalResult, Name, NumGlyphs, Owner, Parent, ParentFont, ParentShowHint, PopupMenu, ShowHint, Showing, Spacing, Style, TabOrder, TabStop, Tag, Top, Visible, Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, Click, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp

TBITMAP

Descrição

Esse objeto armazena um gráfico no formato Bitmap.

Unit

Na VCL:

Graphics

Na CLX:

```
QGraphics
```

Principais Propriedades

```
Canvas, Empty, Format, Handle, Height, Monochrome, Palette, PixelFormat, ScanLine, Transparent,
TransparentColor, TransparentMode  e Width
```

Principais Métodos

```
Assign, ClassName, ClassParent, ClassType, Create, Destroy, Free, LoadFromFile, ReleaseHandle,
ReleasePalette e SaveToFile
```

Principais Eventos

```
OnChange, OnProgress
```

TBlobField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um conjunto arbitrário de bytes cujo tamanho não é predefinido.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsString, BlobSize, BlobType, Calculated,
CanModify, DataSet, DataSize, DataType, DisplayLabel, DisplayName, DisplayText, DisplayWidth,
EditMask, EditMaskPtr, FieldName, FieldNo, GraphicHeader, Index, IsIndexField, IsNull, Name, Owner,
ReadOnly, Required, Size, Tag, Text e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar, LoadFromFile, LoadFromStream,
SaveToFile, SaveToStream e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate
```

TBlobStream

Descrição

Esse objeto fornece uma maneira simples de acessar e modificar um campo do tipo TBlobField, TBytesField ou TVarBytesField lendo ou escrevendo nele como se fosse uma stream ou um arquivo.

Unit

Na VCL:

```
DBTables
```

Principais Propriedades

```
Position, Size
```

Principais Métodos

```
ClassName, ClassParent, ClassType, Create, Destroy, Free, Read, Seek, Truncate e Write
```

Principais Eventos

Esse componente não possui eventos associados.

TBooleanField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um valor booleano.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsSQLTimeStamp, AsString, Calculated, CanModify,
DataSet, DataSize, DataType, DisplayLabel, DisplayName, DisplayText, DisplayWisth, EditFormat,
EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, Name, Owner, ReadOnly,
Required, Size, Tag, Text, Value e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate
```

TBRUSH

Descrição

Esse objeto armazena um pincel para o preenchimento de figuras sólidas, como retângulos e elipses.

Unit

Na VCL:

```
Graphics
```

Na CLX:

```
QGraphics
```

Principais Propriedades

```
Bitmap, Color, Handle e Style
```

Principais Métodos

```
Assign, ClassName, ClassParent, ClassType, Create, Destroy e Free
```

Principais Eventos

```
OnChange
```

TBUTTON

Descrição

O controle TButton representa um botão comum na sua face.

Unit

Na VCL:

```
StdCtrls
```

Na CLX:

```
QStdCtrls
```

Principais Propriedades

```
Align, BoundsRect, Cancel, Caption, ComponentIndex, Cursor, Default, DragCursor, DragMode, Enabled,
Font, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, ModalResult, Name, Owner, Parent,
ParentFont, ParentShowHint, PopupMenu, ShowHint, Showing, Spacing, Style, TabOrder, TabStop, Tag,
Top, Visible, Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, Click, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf,
GetTextLen, Hide, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds,
SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

OnClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp

TBytesField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um conjunto arbitrário de bytes cujo tamanho não é predefinido.

Unit

Na VCL e na CLX:

DB

Principais Propriedades

Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsString, Calculated, CanModify, DataSet, DataSize, DataType, DisplayLabel, DisplayName, DisplayText, DisplayWisth, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, Name, Owner, ReadOnly, Required, Size, Tag, Text e Visible

Principais Métodos

Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData

Principais Eventos

OnChange, OnGetText, OnSetText e OnValidate

TCanvas

Descrição

Esse objeto define uma superfície de desenho.

Unit

Na VCL:

Graphics

Na CLX:

QGraphics

Principais Propriedades

Brush, ClipRect, CopyMode, Font, Handle, LockCount, Pen, PenPos, Pixels, StartCount, TextAlign

Principais Métodos

Arc, BrushCopy, Chord, ClassName, ClassParent, ClassType, CopyRect, Create, Destroy, Draw, DrawFocusRect, Ellipse, FillRect, FloodFill, FrameRect, Free, LineTo, MoveTo, Pie, Polygon, Polyline, Rectangle, RoudRect, StretchDraw, TextHeight, TextOut, TextRect e TextWidth

Principais Eventos

OnChange e OnChanging

TCanvasReport

Unit

Na VCL:

RpCanvas

Na CLX:

QRpCanvas

Descrição

Esta classe é usada para definir métodos a serem usados pelos componentes do Rave Reports que desenham em um Canvas.

Principais Propriedades

Aborted, AccuracyMethod, AscentHeight, Bins, BKColor, old, BottomWaste, BoxLineColor, Canvas, Collate, ColumnEnd, ColumnLinesLeft, ColumnNum, Columns, ColumnStart, ColumnWidth, Copies, CurrentPage, CurrentPass, CursorXPos, CursorYPos, DescentHeight, DeviceName, DevMode, DriverName, Duplex, FileName, FirstPage, FontAlign, FontBaseline, FontBottom, FontCharset, FontColor, FontHandle, FontHeight, FontName, FontPitch, FontRotation, Fonts, FontSize, FontTop, FontWidth, FrameMode, GridVert, Italic, LastPage, LeftWaste, LineBottom, LineHeight, LineHeightMethod, LineMiddle, LineNum, LinesPerInch, LineTop, MacroData, MarginBottom, MarginLeft, MarginRight, MarginTop, MaxCopies, NoBufferLine, NoNTColorFix, NoPrinterPageHeight, NoPrinterPageWidth, Orientation, OriginX, OriginY, OutputInvalid, OutputName, PageHeight, PageInvalid, PageWidth, Papers, PIVar, Port, PrinterIndex, Printers, Printing, ReportDateTime, RightWaste, ScaleX, ScaleY, SectionBottom, SectionLeft, SectionRight, SectionTop, Selection, ShadowDepth, StatusFormat, StatusLabel, StatusText, Stream, StreamMode, Strikeout, Subscript, Superscript, TabColor, TabJustify, TabShade, TextBKMode, Title, TopWaste, TotalPasses, TransparentBitmaps, TruncateText, Underline, Units, UnitsFactor, XDPI, XPos, YDPI, YPos, Version

Principais Métodos

Abort, AbortPage, AdjustLine, AllowAll, AllowPreviewOnly, AllowPrinterOnly, Arc, AssignFont, BrushCopy, CalcGraphicHeight, CalcGraphicWidth, Chord, ClearAllTabs, ClearColumns, ClearTabs, CopyRect, CR, Create, CreateBrush, CreateFont, CreatePen, CreatePoint, CreateRect, Destroy, DrawFocusRect, Draw, Ellipse, Execute, FillRect, Finish, FinishTabBox, FloodFill, FrameRect, GetMemoLine, GetNextLine, GetTab, GotoFooter, GotoHeader, GotoXY, GraphicFieldToBitmap, Home, LF, LinesLeft, LineTo, Macro, MemoLines, MoveTo, NewColumn, NewLine, NewPage, NoPrinters, Pie, Polygon, Polyline, PopFont, PopPos, PopTabs, Print, PrintBitmap, PrintBitmapRect, PrintBlock, PrintCenter, PrintCharJustify, PrintData, PrintDataStream, PrintFooter, PrintHeader, PrintImageRect, PrintJustify, PrintLeft, PrintLn, PrintMemo, PrintRight, PrintTab, PrintXY, PushFont, PushPos, PushTabs, RecoverPrinter, Rectangle, RegisterGraphic, ReleasePrinter, Reset, ResetLineHeight, ResetPrinter, ResetSection, ResetTabs, RestoreFont, RestorePos, RestoreTabs, ReuseGraphic, RoundRect, SaveFont, SavePos, SaveTabs, SelectBin, SelectPaper, SelectPrinter, SetBrush, SetColumns, SetColumnWidth, SetFont, SetPaperSize, SetPen, SetPIVar, SetTab, SetTopOfPage, ShadeToColor, ShowPrintDialog, ShowPrinterSetupDialog, Start, StretchDraw, SupportBin, SupportCollate, SupportDuplex, SupportOrientation, SupportPaper, Tab, TabEnd, TabStart, TabWidth, TextRect, TextWidth, UnregisterGraphic, UpdateStatus, XD2U, XI2D, XI2U, XU2D, XU2I, YD2I, YD2U, YI2D, YI2U, YU2D, YU2I

Principais Eventos

OnAfterPrint, OnBeforePrint, OnDecodeImage, OnNewColumn, OnNewPage, OnPrint, OnPrintFooter, OnPrintHeader, OnPrintPage

TCHART

Descrição

O controle TChart é utilizado para exibir gráficos nos mais diversos formatos.

Unit

Na VCL:

Chart

Principais Propriedades

AllowPanning, Color, Printing, AllowZoom, Foot, DockClientCount, DockClients, PrintMargins, AnimatedZoom, Frame, PrintResolution, AnimatedZoomSteps, Gradient, RightAxis, AxisVisible, Height3D, ScaleLastPage, BackColor, LeftAxis, Series, BackImage, LeftWall, SeriesHeight3D, BackImageInside, Legend, SeriesList, BottomAxis, MarginBottom, SeriesWidth3D, BackWall, DepthAxis, View3Doptions, BottomWall, MarginLeft, Title, BufferedDisplay, MarginRight, TopAxis, CancelMouse, MarginTop, View3d, Canvas, MaxPointsPerPage, View3dWalls, Chart3dPercent, MaxZOrder, Width3D, ChartBounds, Monochrome, ChartHeight, MonochromePrinting, ChartRect, OriginalCursor, ChartWidth, BackImage, ClipPoints, BackImageMode

Principais Métodos

ActiveSeriesLegend, TeeCreateMetafile, PrintRect, AddSeries, GetLabelsSeries, ReCalcWidthHeight, Assign, GetCursorPos, RemoveAllSeries, BackWallRect, GetRectangle, RemoveSeries, CalcClickedpart, GetWidthHeight, RotateLabel, CalcSize3d, IsFreeSeriesColor, SaveToBitmapFile, CalcSize3dWalls, IsScreenHighColor, SaveToMetafile, CanvasChanged, IsValidDataSource, SaveToMetafileEnh, ChartPrintRect, MarkText, SeriesCount, ChartRegionRect, MaxMarkWidth, SeriesDown, ChartXCenter, MaxTextWidth, SeriesTitleLegend, ChartYCenter, MaxXValue, SeriesUp, CheckDatasource, MaxYValue, SetInternalCanvas, MinXValue, Size3d, CopyToClipBoardBitmap, MinYValue, CopyToClipBoardMetafile,

```
NextPage, NumPages, PreviousPage, ExchangeSeries, Print, FontCanvas, PrintLandscape, UndoZoom,
FormattedLegend, PrintOrientation, XLabelText, FormattedValueLegend, PrintPartial, ZoomRect,
GetASeries, PrintPartialCanvas, ZoomPercent, GetAxisSeries, PrintPartialCanvasToScreen,
GetFreeSeriesColor, PrintPortrait, SaveToChartFile
```

Principais Eventos

```
OnAfterDraw, OnDblClick, OnMouseUp, OnAllowScroll, OnGetAxisLabel, OnPageChange, OnClick,
OnGetLegendPos, OnResize, OnClickAxis, OnGetLegendRect, OnScroll, OnClickBackground,
OnGetLegendText, OnUndoZoom, OnClickLegend, OnGetNextAxisLabel, OnZoom, OnClickSeries, OnMouseDown
```

TCHARTSERIES

Descrição

Essa classe representa uma série de dados a serem exibidos em um gráfico criado com o componente TChart.

Unit

Na VCL:

```
TEengine
```

Principais Propriedades

```
Active, AllowSinglePoint, LinkedSeries, VertAxis, Marks, XLabel, ColorEachPoint, ParentChart,
XLabelsSource, ColorSource, PercentFormat, XValue, Cursor, RecalcOptions, XValues, DataSource,
SeriesColor, YValue, Datasources, ShowInlegend, YValues, DesignMaxPoints, Title, ZOrder, FirstValueIndex,
ValueColor, FunctionType, ValueFormat, HorizAxis, ValueList, LastValueIndex, ValueMarkText
```

Principais Métodos

```
ActiveSeriesLegend, TeeCreateMetafile, PrintRect, AddSeries, GetLabelsSeries, ReCalcWidthHeight,
Assign, GetCursorPos, RemoveAllSeries, BackWallRect, GetRectangle, RemoveSeries, CalcClickedpart,
GetWidthHeight, RotateLabel, CalcSize3d, IsFreeSeriesColor, SaveToBitmapFile, CalcSize3dWalls,
IsScreenHighColor, SaveToMetafile, CanvasChanged, IsValidDataSource, SaveToMetafileEnh,
ChartPrintRect, MarkText, SeriesCount, ChartRegionRect, MaxMarkWidth, SeriesDown, ChartXCenter,
MaxTextWidth, SeriesTitleLegend, ChartYCenter, MaxXValue, SeriesUp, CheckDatasource, MaxYValue,
SetInternalCanvas, MinXValue, Size3d, CopyToClipBoardBitmap, MinYValue, CopyToClipBoardMetafile,
NextPage, NumPages, PreviousPage, ExchangeSeries, Print, FontCanvas, PrintLandscape, UndoZoom,
FormattedLegend, PrintOrientation, XLabelText, FormattedValueLegend, PrintPartial, ZoomRect,
GetASeries, PrintPartialCanvas, ZoomPercent, GetAxisSeries, PrintPartialCanvasToScreen,
GetFreeSeriesColor, PrintPortrait, SaveToChartFile
```

Principais Eventos

```
OnAfterDraw, OnDblClick, OnMouseUp, OnAllowScroll, OnGetAxisLabel, OnPageChange, OnClick,
OnGetLegendPos, OnResize, OnClickAxis, OnGetLegendRect, OnScroll, OnClickBackground,
OnGetLegendText, OnUndoZoom, OnClickLegend, OnGetNextAxisLabel, OnZoom, OnClickSeries, OnMouseDown
```

TCHECKBOX

Descrição

O controle TCheckBox é um botão de opção não-exclusiva, que permite ao usuário selecionar ou não uma opção no aplicativo.

Unit

Na VCL:

```
StdCtrls
```

Na CLX:

```
QStdCtrls
```

Principais Propriedades

```
Align, Alignment, AllowGrayed, Caption, Checked, Color, ComponentIndex, Ctl3D, Cursor, DragCursor, DragMode,
Enabled, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Owner, Parent, ParentColor,
ParentFont, ParentShowHint, PopupMenu, ShowHint, Showing, State, TabOrder, TabStop, Tag, Top, Visible, Width
```

Principais Métodos

BeginDrag, BringToFront, CanFocus, ClientToScreen, Dragging, EndDrag, FindComponent, Focused, GetTextBuf, GetTextLen, Hide, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetTextBuf, Show e Update

Principais Eventos

OnClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp

TClientDataSet

Descrição

Esse componente representa uma tabela de forma independente (não associada diretamente a um banco de dados), sendo utilizado em aplicações multicamadas de acesso a bancos de dados ou em aplicações que armazenam dados em arquivos sem utilizar o Activex Data Objects ou o Borland Database Engine.

Unit

Na VCL e na CLX:

DBclient

Principais Propriedades

Active, ActiveAggs, Aggregates, AggregatesActive, CanModify, ChangeCount, Data, DataSetField, DataSize, DataSource, Delta, FetchOnDemand, FileName, Filter, Filtered, FilterOptions, GroupingLevel, HasProvider, IndexDefs, IndexFieldCount, IndexFieldNames, IndexFields, IndexName, KeyExclusive, KeyFieldCount, KeySize, LogChanges, MasterFields, MasterSource, PacketRecords, Params, Provider, ProviderName, ReadOnly, RecNo, RecordCount, RecordSize, RemoteServer, SavePoint, StatusFilter, StoreDefs, AggFields, AutoCalcFields, BlockReadSize, Bof, Bookmark, DefaultFields, Designer, Eof, FieldCount, FieldDefList, FieldDefs, FieldList, Fields, FieldValues, Found, Modified, Name, ObjectView, SparseArrays, State

Principais Métodos

AddIndex, AppendData, ApplyRange, ApplyUpdates, BookmarkValid, Cancel, CancelRange, CancelUpdates, CloneCursor, CompareBookmarks, ConstraintsDisabled, Create, CreateBlobStream, CreateDataSet, DeleteIndex, Destroy, DisableConstraints, EditKey, EditRangeEnd, EditRangeStar, EmptyDataSet, EnableConstraints, FetchBlobs, FetchDetails, FetchParams, FindKey, FindNearest, GetCurrentRecord, GetFieldData, GetGroupState, GetIndexInfo, GetIndexNames, GetNextPacket, GetOptionalParam, GotoCurrent, GotoKey, GotoNearest, LoadFromFile, LoadFromStream, Locate, Lookup, MergeChangeLog, Post, Reconcile, RefreshRecord, RevertRecord, SaveToFile, SaveToStream, SendParams, SetAltRecBuffers, SetKey, SetOptionalParam, SetRange, SetRangeEnd, SetRangeStart, UndoLastChange, UpdateStatus, ActiveBuffer, Append, AppendRecord, CheckBrowseMode, ClearFields, Close, ControlsDisabled, CursorPosChanged, Delete, DisableControls, Edit, EnableControls, FieldByName, FindField, FindFirst, FindLast, FindNext, FindPrior, First, FreeBookmark, GetBlobFieldData, GetBookmark, GetDetailDataSets, GetDetailLinkFields, GetFieldList, GetFieldNames, GetProviderAttributes, GotoBookmark, Insert, InsertRecord, IsEmpty, IsLinkedTo, IsSequenced, Last, MoveBy, Next, Open, Prior, Refresh, Resync, SetFields, Translate, UpdateCursorPos, UpdateRecordAssign, Clear, Close, Create, Destroy, GetAsHandle, GetComponent, HasFormat, Open, SetAsHandle, SetComponent e SetTextBuf

Principais Eventos

OnReconcileError, AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnFilterRecord, OnNewRecord, OnPostError, OnCalcFields, OnDeleteError, OnEditError, OnFilterRecord, OnNewRecord, OnPostError

TClientSocket

Descrição

Esse componente gerencia conexões TCP/IP do lado cliente.

Unit

Na VCL:

scktcomp

Principais Propriedades

```
ClientType, Socket, Active, Address, Host, Port, Service
```

Principais Métodos

```
Create, Destroy, Close e Open
```

Principais Eventos

```
OnConnect, OnConnecting, OnDisconnect, OnError, OnLookup, OnRead, OnWrite
```

TClipboard

Descrição

Esta classe representa o clipboard (área de transferência) do sistema operacional e permite que você transfira textos e gráficos de e para o clipboard.

Unit

Na VCL:

```
Clipbrd
```

Na CLX:

```
QClipbrd
```

Principais Propriedades

```
AsText e Handle
```

Principais Métodos

```
Assign, Clear, Close, Create, Destroy, GetAsHandle, GetComponent, HasFormat, Open, SetAsHandle,
SetComponent e SetTextBuf
```

Principais Eventos

Esse componente não possui eventos associados.

TColorDialog

Descrição

O controle TColorDialog fornece uma caixa de diálogo para a seleção de cores.

Unit

Na VCL:

```
Dialogs
```

Na CLX:

```
QDialogs
```

Principais Propriedades

```
Color, ComponentIndex, Ctl3D, CustomColors, HelpContext, HelpKeyword, HelpTypeHint, ItemHeight,
ItemIndex, Items, Left, Name, Options, Owner, Tag e Top
```

Principais Métodos

```
Execute
```

Principais Eventos

Esse componente não possui eventos associados.

TComboBox

Descrição

O controle TComboBox consiste em uma caixa de edição (na qual o usuário pode digitar um texto) e uma lista de itens que podem ser selecionados.

Unit

Na VCL:

```
StdCtrls
```

Na CLX:

```
QStdCtrls
```

Principais Propriedades

```
Align, BoundsRect, Canvas, Color, ComponentIndex, Ctl3D, Cursor, DragCursor, DragMode,
DropDownCount, Enabled, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, ItemHeight, ItemIndex,
Items, Left, MaxLength, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint,
PopupMenu, SelLengh, SelStart, SelText, ShowHint, Showing, Sorted, Style, TabOrder, TabStop, Tag,
Text, Top, Visible, Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, Clear, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf,
GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SelectAll,
SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnDrawItem, OnDropDown, OnEndDrag, OnEnter,
OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMeasureItem
```

TCOMPONENT

Descrição

A classe Tcomponent é a classe-base de todos os componentes da VCL do Delphi.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DesignInfo,
Name, Owner, Tag e VCLComObject
```

Principais Métodos

```
_AddRef, _Release, AfterConstruction, Assign, BeforeDestruction, ChangeName, ClassInfo, ClassName,
ClassNameIs, ClassParent, ClassType, CleanupInstance, Create, DefaultHandler, DefineProperties,
Destroy, DestroyComponents, Destroying, Dispatch, ExecuteAction, FieldAddress, FindComponent, Free,
FreeInstance, FreeNotification, FreeOnRelease, GetChildOwner, GetChildParent, GetChildren,
GetIDsOfNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetOwner,
GetParentComponent, GetTypeInfo, GetTypeInfoCount, HasParent, InheritsFrom, InitInstance,
InsertComponent, InstanceSize, Invoke, Loaded, MethodAddress, MethodName, NewInstance, Notification,
QueryInterface, ReadState, RemoveComponent, SafeCallException, SetAncestor, SetChildOrder,
SetDesigning, SetInline, SetName, SetParentComponent, UpdateAction, Updated, UpdateRegistry,
Updating, ValidateContainer, ValidateInsert, ValidateRename, WriteState
```

Principais Eventos

esta classe não possui eventos associados

TCONTROLSCROLLBAR

Descrição

Esse objeto define as propriedades HorzScrollBar e VertScrollBar de componentes do tipo TForm e TScrollBox.

Unit

Na VCL:

```
Forms
```

Na CLX:

```
QForms
```

Principais Propriedades

```
Align, ComponentIndex, Increment, Kind, Margin, Name, Owner, Position, Range, ScrollPos, Tag e Visible
```

Principais Métodos

Esse objeto não possui métodos associados.

Principais Eventos

Esse componente não possui eventos associados.

TCorbaConnection

Descrição

Esse componente é responsável por conectar uma aplicação-cliente CORBA a um servidor remoto em uma aplicação multicamada.

Unit

```
corbacon
```

Principais Propriedades

```
AppServer, Cancelable, Connected, HostName, ObjectName, RepositoryID
```

Principais Métodos

```
Create, GetProvider, Destroy
```

Principais Eventos

```
OnCancel, AfterConnect, AfterDisconnect, BeforeConnect, BeforeDisconnect
```

TCurrencyField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um número no formato binário com 15 ou 16 dígitos de precisão, variando de 5.0 * 10-324 a 1.7 * 10+308.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsSQLTimeStamp, AsString, Calculated, CanModify,
Currency, DataSet, DataSize, DataType, DisplayFormat, DisplayLabel, DisplayName, DisplayText,
DisplayWisth, EditFormat, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull,
MaxValue, MinValue, Name, Owner, Precision, ReadOnly, Required, Size, Tag, Text, Value e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate
```

TCustomADODataSet

Descrição

Esta classe permite acesso direto ou via declarações sql a tabelas de bancos de dados através do Mecanismo Activex Data Objects (ADO), sendo a classe-base dos componentes TADODataset, TADOTable, TADOQuery e TADOStoredproc.

Unit

AdoDB

Principais Propriedades

Active, ActiveRecord, AggFields, AutoCalcFields, BlobFieldCount, BlockReadSize, Bof, Bookmark, BookmarkSize, BufferCount, Buffers, CacheSize, CalcBuffer, CalcFieldsSize, CanModify, CommandText, CommandTimeout, CommandType, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Connection, ConnectionString, Constraints, CurrentRecord, CursorLocation, CursorType, DataSetField, DefaultFields, Designer, DesignerData, DesignInfo, Eof, ExecuteOptions, FieldCount, FieldDefList, FieldDefs, FieldList, FieldNoOfs, Fields, FieldValues, Filter, Filtered, FilterGroup, FilterOptions, Found, IndexFieldCount, IndexFields, InternalCalcFields, LockType, MarshalOptions, MaxRecords, Modified, Name, NestedDataSetClass, NestedDataSets, ObjectView, Owner, ParamCheck, Parameters, Prepared, Properties, RecNo, RecordCount, RecordSet, RecordSetState, RecordSize, RecordStatus, Reserved, Sort, SparseArrays, State, Tag, VCLComObject,

Principais Métodos

ActiveBuffer, AfterConstruction, Append, AppendRecord, Assign, BeforeDestruction, BookmarkValid, Cancel, CancelBatch, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Clone, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CursorPosChanged, DefaultHandler, Delete, DeleteRecords, Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EnableControls, ExecuteAction, FieldAddress, FieldByName, FilterOnBookmarks, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GotoBookmark, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsLinkedTo, IsSequenced. Last, LoadFromFile, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, NextRecordset, Open, Post, Prior, Refresh, RemoveComponent, Resync, SafeCallException, SaveToFile, Seek, SetFields, Supports, Translate, UpdateAction, UpdateBatch, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnEndOfRecordset, OnFetchComplete, OnFetchProgress, OnFieldChangeComplete, OnFilterRecord, OnMoveComplete, OnNewRecord, OnPostError, OnRecordChangeComplete, OnRecordsetChangeComplete, OnWillChangeField, OnWillChangeRecord, OnWillChangeRecordset, OnWillMove

TDATABASE

Descrição

Esse componente permite o acesso a um banco de dados. Caso a aplicação não possua um componente desse tipo, mas precise acessar uma tabela de um banco de dados, o Delphi criará um componente TDataBase temporário.

Unit

Na VCL:

DBTables

Principais Propriedades

AliasName, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Connected, DatabaseName, DatasetCount, DataSets, DesignInfo, DriverName, Handle, Isolation, IsSQLBased, KeepConnection, Locale, LoginPrompt, Name, Owner, Params, Tag, Temporary e VCLComObject

Principais Métodos

Close, CloseDataSets, Commit, Open, Rollback e StartTransaction

Principais Eventos

```
AfterConnect, AfterDisconnect, BeforeConnect, BeforeDisconnect, OnLogin
```

TDatasetPageProducer

Descrição

Esse componente gera uma string de comandos HTML de um template com os dados provenientes de um dataset.

Unit

Na VCL e na CLX:

```
DSProd
```

Principais Propriedades

```
DataSet, HTMLFile, HTMLDoc
```

Principais Métodos

```
Content, ContentFromStream, ContentFromString, Create, Destroy
```

Principais Eventos

```
OnHTMLTag
```

TDatasetProvider

Descrição

Esse componente é responsável por fornecer os dados à aplicação-cliente baseado em um dataset.

Unit

Na VCL e na CLX:

```
provider
```

Principais Propriedades

```
DataSet, UpdateMode, Options, Resolver, Constraints, Data, Provider
```

Principais Métodos

```
ApplyUpdates, Create, CreateDataPacket, CreateResolver, Destroy, DoGetDataSet, DoGetParams,
FetchData, GetDataSet, GetRecords, LocateRecord, Notification, Reset, SetDataSet, UpdateRecord,
SetParams, DataRequest, GetMetaData
```

Principais Eventos

```
OnGetDataSetProperties, AfterUpdateRecord, BeforeUpdateRecord, OnGetData, OnUpdateData,
OnUpdateError, OnDataRequest
```

TDatasetTableProducer

Descrição

Esse componente gera uma string de comandos HTML que forma uma tabela com os dados provenientes de um Dataset.

Unit

Na VCL e na CLX:

```
DBWeb
```

Principais Propriedades

```
DataSet, Caption, CaptionAlignment, Columns, Editor, Footer, Header, MaxRows, RowAttributes e TableAttributes
```

Principais Métodos
Content, BeginUpdate, Create, Destroy, EndUpdate

Principais Eventos
OnCreateContent, OnFormatCell, OnGetTableCaption

TDataSource

Descrição

Esse componente estabelece a conexão entre componentes de acesso a bancos de dados e controles que exibem os valores de campos de bancos de dados.

Unit

Na VCL e na CLX:

DB

Principais Propriedades
AutoEdit, DataSet, Enabled, Name, Owner, State e Tag

Principais Métodos
Create, Destroy, Edit e Free

Principais Eventos
OnDataChange, OnStateChange e OnUpdateChange

TDateField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena uma data.

Unit

Na VCL e na CLX:

DB

Principais Propriedades
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsSQLTimeStamp, AsString, Calculated, CanModify, DataSet, DataSize, DataType, DisplayLabel, DisplayName, DisplayText, DisplayWidth, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, Name, Owner, ReadOnly, Required, Size, Tag, Text e Visible

Principais Métodos
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData

Principais Eventos
OnChange, OnGetText, OnSetText e OnValidate

TDateTimeField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena uma data e uma hora.

Unit

Na VCL e na CLX:

DB

Principais Propriedades

Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsString, Calculated, CanModify, DataSet, DataSize, DataType, DisplayFormat, DisplayLabel, DisplayName, DisplayText, DisplayWidth, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, Name, Owner, ReadOnly, Required, Size, Tag, Text, Value e Visible

Principais Métodos

Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData

Principais Eventos

OnChange, OnGetText, OnSetText e OnValidate

TDBChart

Descrição

O controle TChart é utilizado para exibir gráficos nos mais diversos formatos.

Unit

Na VCL:

Chart

Principais Propriedades

AllowPanning, AutoRefresh, RefreshInterval, DockClientCount, DockClients, ShowGlassCursor, Color, Printing, AllowZoom, Foot, PrintMargins, AnimatedZoom, Frame, PrintResolution, AnimatedZoomSteps, Gradient, RightAxis, AxisVisible, Height3D, ScaleLastPage, BackColor, LeftAxis, Series, BackImage, LeftWall, SeriesHeight3D, BackImageInside, Legend, SeriesList, BottomAxis, MarginBottom, SeriesWidth3D, BackWall, DepthAxis, View3Doptions, BottomWall, MarginLeft, Title, BufferedDisplay, MarginRight, TopAxis, CancelMouse, MarginTop, View3d, Canvas, MaxPointsPerPage, View3dWalls, Chart3dPercent, MaxZOrder, Width3D, ChartBounds, Monochrome, ChartHeight, MonochromePrinting, ChartRect, OriginalCursor, ChartWidth, BackImage, ClipPoints, BackImageMode

Principais Métodos

ActiveSeriesLegend, TeeCreateMetafile, PrintRect, CheckDataSource, IsValidDataSource, RefreshData, RefreshDataSet, AddSeries, GetLabelsSeries, ReCalcWidthHeight, Assign, GetCursorPos, RemoveAllSeries, BackWallRect, GetRectangle, RemoveSeries, CalcClickedpart, GetWidthHeight, RotateLabel, CalcSize3d, IsFreeSeriesColor, SaveToBitmapFile, CalcSize3dWalls, IsScreenHighColor, SaveToMetafile, CanvasChanged, IsValidDataSource, SaveToMetafileEnh, ChartPrintRect, MarkText, SeriesCount, ChartRegionRect, MaxMarkWidth, SeriesDown, ChartXCenter, MaxTextWidth, SeriesTitleLegend, ChartYCenter, MaxXValue, SeriesUp, CheckDatasource, MaxYValue, SetInternalCanvas, MinXValue, Size3d, CopyToClipBoardBitmap, MinYValue, CopyToClipBoardMetafile, NextPage, NumPages, PreviousPage, ExchangeSeries, Print, FontCanvas, PrintLandscape, UndoZoom, FormattedLegend, PrintOrientation, XLabelText, FormattedValueLegend, PrintPartial, ZoomRect, GetASeries, PrintPartialCanvas, ZoomPercent, GetAxisSeries, PrintPartialCanvasToScreen, GetFreeSeriesColor, PrintPortrait, SaveToChartFile

Principais Eventos

OnAfterDraw, OnDblClick, OnMouseUp, OnAllowScroll, OnGetAxisLabel, OnPageChange, OnClick, OnGetLegendPos, OnResize, OnClickAxis, OnGetLegendRect, OnScroll, OnClickBackground, OnGetLegendText, OnUndoZoom, OnClickLegend, OnGetNextAxisLabel, OnZoom, OnClickSeries, OnMouseDown, OnProcessRecords

TDBCheckBox

Descrição

O controle TDBCheckBox é um botão de opção não-exclusiva que permite ao usuário selecionar ou não uma opção no aplicativo. Esse controle é semelhante a TCheckBox, exceto que o seu valor está associado a um campo de um registro de um banco de dados. Quando o controle estiver selecionado, então a string armazenada em sua propriedade ValueChecked será igual à armazenada no campo correspondente do banco de dados. Se o controle não estiver selecionado, então a string armazenada em sua propriedade ValueUnChecked será igual à armazenada no campo correspondente do banco de dados.

Unit

Na VCL:

```
DBCtrls
```

Na CLX:

```
QDBCtrls
```

Principais Propriedades

```
Align, Alignment, AllowGrayed, Caption, Checked, Color, ComponentIndex, Ctl3D, Cursor, DataField,
DataSource, DragCursor, DragMode, Enabled, Font, Height, HelpContext, HelpKeyword, HelpTypeHint,
Left, Name, Owner, Parent, ParentColor, ParentFont, ParentShowHint, PopupMenu, ReadOnly, ShowHint,
Showing, State, TabOrder, TabStop, Tag, Top, ValueChecked, ValueUnchecked, Visible, Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf,
GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack,
SetBounds, SetTextBuf, Show e Update
```

Principais Eventos

```
OnClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp,
OnMouseDown, OnMouseMove e OnMouseUp
```

TDBCOMBOBOX

Descrição

O controle TDBComboBox consiste em uma caixa de edição (na qual o usuário pode digitar um texto) e uma lista de itens que podem ser selecionados. Esse controle é semelhante a TComboBox, exceto que o seu valor está associado a um campo de um registro de um banco de dados. Quando o usuário digitar ou selecionar um item, a string correspondente será armazenada no campo respectivo do banco de dados.

Unit

Na VCL:

```
DBCtrls
```

Na CLX:

```
QDBCtrls
```

Principais Propriedades

```
Align, BoundsRect, Color, ComponentIndex, Ctl3D, Cursor, DataField, DataSource, DragCursor,
DragMode, DropDownCount, Enabled, Fields, Font, Handle, Height, HelpContext, HelpKeyword,
HelpTypeHint, ItemHeight, ItemIndex, Items, Left, Name, Owner, Parent, ParentColor, ParentCtl3D,
ParentFont, ParentShowHint, PopupMenu, ReadOnly, SelLengh, SelStart, SelText, ShowHint, Showing,
Sorted, Style, TabOrder, TabStop, Tag, Text, Top, Visible, Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, Clear, ClientToScreen, CopyToClipBoard, CutToClipBoard, Dragging,
EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Invalidate, PasteFromClipBoard, Refresh, Repaint,
ScaleBy, ScreenToClient, ScrollBy, SelectAll, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnDrawItem, OnDropDown, OnEndDrag, OnEnter,
OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMeasureItem
```

TDBCTRLGRID

Descrição

Esse controle é semelhante ao controle TDBGrid, com a diferença de que este permite que se controle o layout e a aparência de cada registro de um banco de dados exibido em uma grade. Sua aplicação pode utilizar a grade para exibir, inserir, deletar ou editar campos do banco de dados associado.

Unit

Na VCL:

```
DBGrids
```

Principais Propriedades

```
AllowDelete, AllowInsert, Canvas, ColCount, Color, Cursor, DataSource, DragCursor, DragMode, EditMode,
Font, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Orientation, PanelBorder,
PanelCount, PanelHeight, PanelIndex, PanelWidth, ParentColor, ParentFont, ParentShowHint, PopupMenu,
RowCount, ShowFocus, ShowHint, TabOrder, TabStop, Tag, Top, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, ClientToScreen, Dragging, EndDrag, FindComponent, Focused,
GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy,
SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnColEnter, OnColExit, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown,
OnKeyPress, OnKeyUp
```

TDBEdit

Descrição

O controle TDBEdit consiste em uma caixa de edição (na qual o usuário pode digitar um texto). Esse controle é semelhante ao TEdit, exceto que o seu valor está associado a um campo de um registro de um banco de dados. Quando o usuário digitar um texto na caixa de edição, a string correspondente será armazenada no campo respectivo do banco de dados.

Unit

Na VCL:

```
DBCtrls
```

Na CLX:

```
QDBCtrls
```

Principais Propriedades

```
Align, AutoSelect, AutoSize, BorderStyle, BoundsRect, CharCase, Color, ComponentIndex, Ctl3D,
Cursor, DataField, DataSource, DragCursor, DragMode, EditText, Enabled, Fields, Font, Height,
HelpContext, HelpKeyword, HelpTypeHint, IsMasked, Left, MaxLength, Modified, Name, Owner, Parent,
ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PassWordChar, PopupMenu, ReadOnly, SelLengh,
SelStart, SelText, ShowHint, Showing, Style, TabOrder, TabStop, Tag, Text, Top, Visible, Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, Clear, ClearSelection, ClientToScreen, CopyToClipBoard,
CutToClipBoard, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Invalidate,
PasteFromClipBoard, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SelectAll, SendToBack,
SetBounds, SetFocus, SetSelTextBuf, SetTextBuf, Show, Update e ValidateEdit
```

Principais Eventos

```
OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown,
OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp
```

TDBGrid

Descrição

Esse controle permite acesso aos dados de um banco de dados e exibe-os em uma grade. Sua aplicação pode utilizar a grade para exibir, inserir, deletar ou editar campos do banco de dados associado.

Unit

Na VCL:

```
DBGrids
```

Na CLX:

```
QDBGrids
```

Principais Propriedades

```
Align, BorderStyle, BoundsRect, Brush, Canvas, ClientHeight, ClientOrigin, ClientRect, ClientWidth,
Color, ComponentIndex, Ctl3D, Cursor, DataSource, DefaultDrawing, i :=
Panel1.DockClientCount;DragCursor, DragMode, EditorMode, Enabled, Fields, FixedColor, Font, Height,
HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Options, Owner, Parent, ParentColor,
ParentCtl3D, ParentFont, PopupMenu, ReadOnly, SelectedField, SelectedIndex, Showing, TabOrder,
TabStop, Tag, Top, TopRow, Visible, Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, ClientToScreen, Dragging, EndDrag, FindComponent, Focused,
GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy,
SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnColEnter, OnColExit, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown,
OnKeyPress, OnKeyUp
```

TDBIMAGE

Descrição

Esse controle permite exibir uma imagem armazenada em um campo de um registro de um banco de dados como um objeto BLOB (Bynary Large Object).

Unit

Na VCL:

```
DBCtrls
```

Na CLX:

```
QDBCtrls
```

Principais Propriedades

```
Align, AutoDisplay, BorderStyle, Center, Color, ComponentIndex, Ctl3D, Cursor, DataField,
DataSource, DragCursor, DragMode, Enabled, Fields, Font, Handle, Height, HelpContext, HelpKeyword,
HelpTypeHint, Left, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint,
PopupMenu, ReadOnly, ShowHint, Stretch, TabOrder, TabStop, Tag, Top, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, ClientToScreen, CopyToClipboard, CutToClipboard, Dragging, EndDrag,
Focused, Hide, Invalidate, LoadPicture, PasteFromClipBoard, Refresh, Repaint, ScreenToClient,
SendToBack, SetBounds, Show e Update
```

Principais Eventos

```
OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress,
OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp
```

TDBLISTBOX

Descrição

O controle TDBListBox consiste em uma caixa de listagem. Esse controle é semelhante a TListBox, exceto que o seu valor está associado a um campo de um registro de um banco de dados. Quando o usuário selecionar um item da caixa de listagem, a string correspondente será armazenada no campo respectivo do banco de dados.

Unit

Na VCL:

```
DBCtrls
```

Na CLX:

QDBCtrls

Principais Propriedades

Align, BorderStyle, BoundsRect, Brush, Canvas, Color, ComponentIndex, Ctl3D, Cursor, DataField, DataSource, DragCursor, DragMode, Enabled, Fields, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, ItemIndex, IntegralHeight, ItemHeight, Items, Left, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, PopupMenu, ReadOnly, SelCount, Selected, Sorted, Style, TabOrder, TabStop, Tag, Top, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, Clear, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, ItemAtPos, ItemRect, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnClick, OnDblClick, OnDragDrop, OnDragOver, OnDrawItem, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMeasureItem, OnMouseDown, OnMouseMove e OnMouseUp

TDBLookupComboBox

Descrição

O controle TDBLookupComboBox consiste em uma caixa combo que está associada aos campos de um banco de dados.

Unit

Na VCL:

DBCtrls

Na CLX:

QDBCtrls

Principais Propriedades

Color, Ctl3D, Cursor, DataField, DataSource, DragCursor, DragMode, DropDownAlign, DropDownRows, DropDownWidth, Enabled, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, KeyField, KeyValue, Left, ListField, ListSource, ListVisible, Name, Owner, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu, ShowHint, TabOrder, TabStop, Text e Visible

Principais Métodos

CloseUp e DropDown

Principais Eventos

OnClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp e OnStartDrag

TDBLookupListBox

Descrição

O controle TDBLookupListBox consiste em uma caixa de listagem que está associada aos campos de um banco de dados. Esse controle é semelhante a TListBox, exceto que o seu valor está associado a um campo de um registro de um banco de dados. Deve ser usado em programas desenvolvidos para os ambientes Windows 95 ou Windows NT em lugar de TDBLookupList.

Unit

Na VCL:

DBCtrls

Na CLX:

QDBCtrls

Principais Propriedades

Align, BorderStyle, BoundsRect, Color, ComponentIndex, Ctl3D, Cursor, DataField, DataSource, DragCursor, DragMode, Enabled, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu, ReadOnly, ShowHint, TabOrder, TabStop, Tag, Top, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, ClientToScreen, Create, DefaultDrawColumnCell, Dragging, EndDrag, FindComponent, Focused, Free, GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show, Update e ValidFieldIndex

Principais Eventos

OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp e OnStartDrag

TDBMEMO

Descrição

O controle TDBMemo exibe texto para o usuário e permite que o ele exiba e digite dados no controle. Esse controle é semelhante a TMemo, exceto que o seu valor está associado a um campo de um registro de um banco de dados. Quando o usuário digitar um texto no controle, esse texto é armazenado no campo respectivo do banco de dados.

Unit

Na VCL:

DBCtrls

Na CLX:

QDBCtrls

Principais Propriedades

Align, Alignment, AutoDisplay, BorderStyle, BoundsRect, Color, ComponentIndex, Ctl3D, Cursor, DataField, DataSource, DragCursor, DragMode, Enabled, Fields, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Lines, MaxLength, Modified, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu, ReadOnly, ScrollBars, SelLengh, SelStart, SelText, ShowHint, Showing, TabOrder, TabStop, Tag, Text, Top, Visible, WantTabs, Width e WordWrap

Principais Métodos

BeginDrag, BringToFront, CanFocus, Clear, ClientToScreen, Dragging, EndDrag, Focused, GetSelTextBuf, GetTextBuf, GetTextLen, Hide, Invalidate, LoadMemo, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SelectAll, SendToBack, SetBounds, SetFocus, SetSelTextBuf, SetTextBuf, Show, Update e ValidateEdit

Principais Eventos

OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp

TDBMEMOBUF

Unit

Na VCL:

RpDBUtil Na CLX:

QRpDBUtil

Descrição

Adiciona processamento de campos do tipo Memo em bancos de dados à classe TMemoBuf.

Principais Propriedades

BaseReport, Buffer, BufferInc, Field, Justify, MaxSize, Memo, NoCRLF, NoNewLine, Pos, PrintEnd, PrintStart, RichEdit, RTFField, RTFText, Size, Text, Version

Principais Métodos

Append, AppendMemoBuf, ConstraintHeightLeft, Delete, Empty, FreeSaved, InsertMemoBuf, Insert, LoadFromFile, LoadFromStream, MemoHeightLeft, MemoLinesLeft, PrintHeight, PrintLines, ReplaceAll, Reset, RestoreBuffer, RestoreState, RTFLoadFromFile, RTFLoadFromStream, SaveBuffer, SaveState, SaveToStream, SearchFirst, SearchNext, SetData

Principais Eventos

Esta classe não tem eventos associados.

TDBNavigator

Descrição

O controle TDBNavigator permite que você se mova pelos campos de um banco de dados por meio de componentes do tipo TTable e TQuery, e realize operações sobre o banco de dados, como inserir um registro em branco ou posicionar-se sobre um registro. Normalmente esse controle é empregado simultaneamente a outros controles que permitem a edição e exibição dos dados.

Unit

Na VCL:

DBCtrls

Na CLX:

QDBCtrls

Principais Propriedades

Align, BoundsRect, ComponentIndex, ConfirmDelete, Ctl3D, Cursor, DataSource, DragCursor, DragMode, Height, HelpContext, HelpKeyword, HelpTypeHint, Hints, Left, Name, Owner, Parent, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu, Showing, ShowHint, TabOrder, TabStop, Tag, Top, Visible, VisibleButtons e Width

Principais Métodos

BeginDrag, BringToFront, BtnClick, CanFocus, Clear, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnMouseDown, OnMouseMove, OnMouseUp e OnResize

TDBRadioGroup

Descrição

O controle TDBRadioGroup permite a exibição de um conjunto de botões de rádio que representam opções mutuamente exclusivas. Esse controle é semelhante a TRadioGroup, exceto que, nesse caso, as opções representadas pelos botões de rádio estão associadas a campos de registros de um banco de dados.

Unit

Na VCL:

DBCtrls

Na CLX:

QDBCtrls

Principais Propriedades

Align, Caption, Color, Columns, ComponentIndex, Ctl3D, Cursor, DataField, DataSource, DragCursor, DragMode, Enabled, Fields, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, ItemIndex, Items, Left, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu, ReadOnly, ShowHint, Showing, TabOrder, TabStop, Tag, Top, Value, Values, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, ClientToScreen, ContainsControl, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter e OnExit

TDBRICHEDIT

Descrição

O controle TDBRichEdit é semelhante ao controle TMemo, mas permite a aplicação de fontes com diferentes atributos a partes distintas do texto inserido no controle.

Unit

Na VCL:

DBCtrls

Principais Propriedades

Align, Alignment, BorderStyle, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect, ClientWidth, Color, ComponentCount, ComponentIndex, Components, ControlCount, Controls, Ctl3D, Cursor, DataSource, DataField, DragMode, Enabled, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeHideScrollBars, HideSelection, Hint, Left, Lines, MaxLength, Name, Owner, Paragraph, Parent, ParentColor, ParentCtl3D, ParentFont, PlainText, PopupMenu, ReadOnly, ScrollBars, SetAttributes, ShowHint, Showing, TabOrder, TabStop, Tag, Top, Visible, WantTabs, WantReturns, Width e WordWrap

Principais Métodos

Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen, ContainsControl, ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindComponent, FindText, Focused, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl, Invalidate, Print, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp, OnProtectChange, OnResizeRequest, OnSaveClipboard, OnSelectionChange e OnStartDrag

TDBTEXT

Descrição

O controle TDBText permite a exibição de um texto em um formulário. Esse controle É semelhante ao TLabel, exceto que, nesse caso, o texto exibido está associado a um campo de um registro de um banco de dados.

Unit

Na VCL:

DBCtrls

Na CLX:

QDBCtrls

Principais Propriedades

Align, Alignment, AutoSize, BoundsRect, Color, ComponentIndex, Cursor, DataField, DataSource, DragCursor, DragMode, Enabled, Fields, Font, Height, Hint, Left, Name, Owner, Parent, ParentColor, ParentFont, ParentShowHint, PopupMenu, ShowHint, Tag, Top, Transparent, Visible, Width e WordWrap

Principais Métodos

BeginDrag, BringToFront, CanFocus, ClientToScreen, ContainsControl, Dragging, EndDrag, GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScreenToClient, SendToBack, SetBounds, SetTextBuf, Show e Update

Principais Eventos
OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnMouseDown, OnMouseMove e OnMouseUp

TDComConnection

Descrição

Esse componente é responsável por conectar uma aplicação-cliente DCOM a um servidor remoto em uma aplicação multicamada.

Unit

Na VCL:

MConnect

Principais Propriedades
ComputerName, Connected, AppServer, LoginPrompt, ObjectBroker, ServerGUID, ServerName

Principais Métodos
Create, DoConnect, GetProvider, Destroy

Principais Eventos
OnGetUserName, OnLogin, AfterConnect, AfterDisconnect, BeforeConnect, BeforeDisconnect

TDDEClientConv

Descrição

Esse componente estabelece uma conversação DDE com uma aplicação servidora.

Unit

Na VCL:

DDEMan

Principais Propriedades
ComponentIndex, ConnectMode, DDEService, DDETopic, FormatChars, Name, Owner, ServiceApplication e Tag

Principais Métodos
CloseLink, ExecuteMacro, ExecuteMacroLines, OpenLink, PokeData, PokeDataLines, RequestData e SetLink

Principais Eventos
OnClose e OnOpen

TDDEClientItem

Descrição

Esse componente define o item da aplicação-cliente em uma conversação DDE.

Unit

Na VCL:

DDEMan

Principais Propriedades
ComponentIndex, DDEConv, DDEItem, Lines, Name, Owner, Tag e Text

Principais Métodos
Esse componente não possui métodos associados.

Principais Eventos
OnChange

TDDEServerConv

Descrição

Esse componente estabelece uma conversação DDE com uma aplicação-cliente.

Unit

Na VCL:

DDEMan

Principais Propriedades

ComponentIndex, Name, Owner e Tag

Principais Métodos

Esse componente não possui métodos associados.

Principais Eventos

OnClose, OnExecuteMacro e OnOpen

TDDEServerItem

Descrição

Esse componente define o item da aplicação servidora em uma conversação DDE.

Unit

Na VCL:

DDEMan

Principais Propriedades

ComponentIndex, Lines, Name, Owner, ServConv, Tag e Text

Principais Métodos

CopyToClipboard

Principais Eventos

OnChange e OnPokeData

TDecisionCube

Descrição

Esse componente permite a exibição de dados na forma tabular.

Unit

Na VCL:

mxdb

Principais Propriedades

Dataset, DimensionMap, MaxCells, MaxDimensions, MaxSummary, Name, ShowProgressDialog e Tag

Principais Métodos

Create, Destroy, GetDetailSQL, GetSQL, ShowCubeDialog

Principais Eventos

AfterClose, AfterOpen, BeforeClose, BeforeOpen, OnLowCapacity, OnRefresh

TDecisionGraph

Descrição

O controle TDecisionGraph é utilizado para exibir gráficos de informações manipuladas pelos componentes DecisionCube.

Unit

Na VCL:

```
Chart
```

Principais Propriedades

```
AllowPanning, Color, Printing, AllowZoom,Foot, PrintMargins, AnimatedZoom, DecisionCube, Frame,
PrintResolution, AnimatedZoomSteps, Gradient, RightAxis, AxisVisible, Height3D, ScaleLastPage, BackColor,
LeftAxis, Series, BackImage, LeftWall, SeriesHeight3D, BackImageInside, Legend, SeriesList, BottomAxis,
MarginBottom, SeriesWidth3D, BackWall, DepthAxis, View3Doptions, BottomWall, MarginLeft, Title,
BufferedDisplay, MarginRight, TopAxis, CancelMouse, MarginTop, View3d, Canvas, MaxPointsPerPage,
View3dWalls, Chart3dPercent, MaxZOrder, Width3D, ChartBounds, Monochrome, ChartHeight, MonochromePrinting,
ChartRect, OriginalCursor, ChartWidth, BackImage, ClipPoints, BackImageMode
```

Principais Métodos

```
ActiveSeriesLegend, TeeCreateMetafile, PrintRect, AddSeries, GetLabelsSeries, ReCalcWidthHeight,
Assign, GetCursorPos, RemoveAllSeries, BackWallRect, GetRectangle, RemoveSeries, CalcClickedpart,
GetWidthHeight, RotateLabel, CalcSize3d, IsFreeSeriesColor, SaveToBitmapFile, CalcSize3dWalls,
IsScreenHighColor, SaveToMetafile, CanvasChanged, IsValidDataSource, SaveToMetafileEnh,
ChartPrintRect, MarkText, SeriesCount, ChartRegionRect, MaxMarkWidth, SeriesDown, ChartXCenter,
MaxTextWidth, SeriesTitleLegend, ChartYCenter, MaxXValue, SeriesUp, CheckDatasource, MaxYValue,
SetInternalCanvas, MinXValue, Size3d, CopyToClipBoardBitmap, MinYValue, CopyToClipBoardMetafile,
NextPage, NumPages, PreviousPage, ExchangeSeries, Print, FontCanvas, PrintLandscape, UndoZoom,
FormattedLegend, PrintOrientation, XLabelText, FormattedValueLegend, PrintPartial, ZoomRect,
GetASeries, PrintPartialCanvas, ZoomPercent, GetAxisSeries, PrintPartialCanvasToScreen,
GetFreeSeriesColor, PrintPortrait, SaveToChartFile
```

Principais Eventos

```
OnAfterDraw, OnDblClick, OnMouseUp, OnAllowScroll, OnGetAxisLabel, OnPageChange, OnClick,
OnGetLegendPos, OnResize, OnClickAxis, OnGetLegendRect, OnScroll, OnClickBackground,
OnGetLegendText, OnUndoZoom, OnClickLegend, OnGetNextAxisLabel, OnZoom, OnClickSeries, OnMouseDown
```

TDECISIONGRID

Descrição

Esse componente exibe os dados manipulados pelos componentes DecisionCube, DecisionQuery, DecisionSource e DecisionPivot.

Unit

Na VCL:

```
mxgrid
```

Principais Propriedades

```
CaptionColor, CaptionFont, Cells, ColCount, DataColor, DataFont, DataSumColor, DecisionSource,
DefaultColWidth, DefaultRowHeight, Dimensions, FixedCols, FixedRows, GridLineColor, GridLineWidth,
LabelColor, LabelFont, LabelSumColor, Options, RowCount, ShowCubeEditor, Totals, BorderStyle,
DefaultDrawing, ScrollBars, BevelEdges, BevelInner, BevelKind, BevelOuter, BevelWidth, Brush,
ClientOrigin, ClientRect, ControlCount, Controls, DockClientCount, DockClients, DoubleBuffered,
Handle, HelpContext, HelpKeyword, HelpTypeParentWindow, Showing, TabOrde, TabStop, Align,
BoundsRect, ClientHeight, ClientWidth, Constraints, ControlState, ControlStyle, Cursor,
DockOrientation, Enabled, Floating, FloatingDockSiteClass, Height, Hint, HostDockSite, Left,
LRDockWidth, Name, Parent, ShowHint, TBDockHeight, Top, UndockHeight, UndockWidth, Visible, Width,
WindowProc, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle,
DesignInfo, Owner, Tag, VCLComObject
```

Principais Métodos

```
CellDrawState, CellValueArray, Create, Destroy, Notification, MouseCoord, CanFocus, ContainsControl,
ControlAtPos, CreateParented, CreateParentedControl, DefaultHandler, DisableAlign, DockDrop,
EnableAlign, FindChildControl, FlipChildren, Focused, GetTabOrderList, HandleAllocated, HandleNeeded,
InsertControl, Invalidate, MouseWheelHandler, PaintTo, Realign, RemoveControl, Repaint, ScaleBy,
ScrollBy, SetBounds, SetFocus, Update, UpdateControlState, BeginDrag, Dock, DrawTextBiDiModeFlags,
DrawTextBiDiModeFlagsReadingOnly, BringToFront, ClientToScreen, DragDrop, Dragging, EndDrag,
GetControlsAlignment, GetParentComponent, GetTextBuf, GetTextLen, HasParent, Hide, InitiateAction,
```

IsRightToLeft, ManualDock, ManualFloat, Perform, Refresh, ReplaceDockedControl, ScreenToClient, SendToBack, SetTextBuf, Show,UseRightToLeftAlignment, UseRightToLeftReading, UseRightToLeftScrollBar, DestroyComponents, Destroying, ExecuteAction, FindComponent, FreeNotification, FreeOnRelease, GetNamePath, InsertComponent, RemoveComponent, SafeCallException, UpdateAction, Assign

Principais Eventos

OnDecisionDrawCell, OnDecisionExamineCell, OnTopLeftChanged, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnMouseDown, OnMouseMove, OnMouseUp, OnResize, OnStartDrag

TDECISIONPIVOT

Descrição

Esse componente define o pivô para um conjunto de componentes DecisionCube.

Unit

Na VCL:

mxpivsrc

Principais Propriedades

BorderStyle, BorderWidth, ButtonAutoSize, ButtonHeight, ButtonSpacing, ButtonWidth, DecisionSource, GroupLayout, Groups, GroupSpacing, Alignment, BevelInner, BevelOuter, BevelWidth, Name, Tag, Top, Width, Height, Left, Cursor, Font, Align, Ctl3D, TabOrder,TabStop, ShowHint, ParentFont, ParentColor, ParentShowHint, ParentCtl3D, PopupMenu, Visible

Principais Métodos

SetBounds, CanFocus, Focused, ContainsControl

Principais Eventos

OnEnter, OnExit, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnResize, OnStartDragTDecisionQuery

TDECISIONQUERY

Descrição

Esse componente é uma especialização do componente TQuery para utilização com o componente TDecisionCube.

Unit

Na VCL:

mxTables

Principais Propriedades

Active, AutoCalcFilelds, BOB, CanModify, Database, DatabaseName, DatabaseSource, DBHandle, DBLocale, EOF, FieldCount, FieldDefs, Field, Handle, Local, Locale, Modified, Name, Owner, ParamCount, Params, Prepared, RecordCount, RequestLive, SQL, SQLBinary, State, StmtHandle, Tag, Text, Unidirecional e UpdateMode

Principais Métodos

Append, AppendRecord, Cancel, CheckBrowseMode, ClearFields, Close, CursorPosChanged, Delete, DisableControls, Edit, EnableControls, ExecSQL, FieldByName, FindField, First, FreeBookmark, GetBookmark, GetFieldNames, GotoBookmark, Insert, InsertRecord, Last, MoveBy, Next, Open, ParamByName, Post, Prepare, Prior, Refresh, SetFields, UnPrepared, UpdateCursorPos e UpdateRecord

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, OnCalcFields e OnNewRecord

TDECISIONSOURCE

Descrição

Esse componente estabelece a conexão entre o componente DecisionCube e os componentes DecisionPivot, DecisionGrid e DecisionGraph.

Unit

Na VCL:

```
mxdb
```

Principais Propriedades

```
ControlType, CurrentSum, DecisionCube, nColDims, nDataCols, nDataRows, nDims, nOpenColDims,
nOpenRowDims, nRowDims, nSums, Ready, SparseCols, SparseRows
```

Principais Métodos

```
CloseDimIndexRight, Create, Destroy, DrillDimIndex, Get2DdataAsVariant, GetActiveDim,
GetDataAsString, GetDataAsVariant, GetDimensionMemberCount, GetDimensionName, GetGroup,
GetGroupExtent, GetIndex, GetMemberAsString, GetMemberAsVariant, GetRowState, GetState,
GetSummaryName, GetValue, GetValueArray, GetValueIndex, MoveDimIndexes, OpenDimIndexLeft,
OpenDimIndexRight, SetCurrentSummary, SwapDimIndexes, ToggleDimIndex
```

Principais Eventos

```
OnAfterPivot, OnBeforePivot, OnLayoutChange, OnNewDimensions, OnStateChange, OnSummaryChange
```

TDirectoryListBox

Descrição

O controle TDirectoryListBox é uma caixa de listagem especial que está relacionada à estrutura de diretórios do drive correntemente selecionado no seu computador, e permite ao usuário mudar o diretório corrente, atribuindo-o à propriedade Directory.

Unit

Na VCL:

```
FileCtrl
```

Principais Propriedades

```
Align, BoundsRect, Color, Columns, ComponentIndex, Ctrl3D, Cursor, Directory, DirLabel, DragCursor, DragMode,
Drive, Enabled, FileList, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, IntegralHeight,
ItemHeight, Left, Name, Owner, Parent, ParentColor, ParentCtrl3D, ParentFont, ParentShowHint, PopupMenu,
Selected, ShowHint, Showing, TabOrder, TabStop, Tag, Top, TopIndex, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, Clear, ClientToScreen, Dragging, EndDrag, Focused, GetItemPath,
GetTextBuf, GetTextLen, Hide, Invalidate, ItemAtPos, ItemRect, Refresh, Repaint, ScaleBy,
ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnClick, OnDblClick, OnDragDrop, OnDragOver, OnDropDown, OnEndDrag, OnEnter, OnExit, OnKeyDown,
OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp
```

TDrawGrid

Descrição

O controle TDrawGrid permite a exibição de um conjunto de dados na forma de um arranjo de linhas e colunas.

Unit

Na VCL:

```
Grids
```

Na CLX:

```
QGrids
```

Principais Propriedades

```
Align, BorderStyle, BoundsRect, Brush, Canvas, Col, ColCount, Color, ColWidthds, ComponentIndex,
Ctrl3D, Cursor, DefaultColWidth, DefaultDrawing, DefaultRowHeight, DragCursor, DragMode, EditorMode,
```

Enabled, FixedColors, FixedCols, FixedRows, GridHeight, GridLineWidth, GridWidth, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, LeftCol, Name, Owner, Parent, ParentColor, ParentCtrl3D, ParentFont, PopupMenu, Row, RowCount, RowHeights, ScrollBars, Selection, Showing, TabOrder, TabStop, TabStops, Tag, Top, TopRow, Visible, VisibleColCount, VisibleRowCount e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, CellRect, ClassName, ClassParent, ClassType, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Invalidate, MouseToCell, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnClick, OnColumnMoved, OnDblClick, OnDragDrop, OnDragOver, OnDrawCell, OnEndDrag, OnEnter, OnExit, OnGetEditMask, OnGetEditText, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp, OnRowMoved, OnSelectCell, OnSetEditText e OnTopLeftChanged

TDRIVECOMBOBOX

Descrição

O controle TDriveComboBox é uma caixa de listagem especial que exibe todos os drives disponíveis quando a aplicação é executada.

Unit

Na VCL:

FileCtrl

Principais Propriedades

Align, BoundsRect, Color, ComponentIndex, Ctrl3D, Cursor, DirList, DragCursor, DragMode, Drive, Enabled, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, Item, ItemIndex, Left, Name, Owner, Parent, ParentColor, ParentCtrl3D, ParentFont, PopupMenu, SelLength, SelStart, SelText, Showing, TabOrder, TabStop, Tag, Text TextCase, Top, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, Clear, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SelectAll, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnDropDown, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress e OnKeyUp

TEDIT

Descrição

O controle TEdit consiste em uma caixa de edição (na qual o usuário pode digitar um texto).

Unit

Na VCL:

StdCtrls

Na CLX:

QStdCtrls

Principais Propriedades

Align, AutoSelect, AutoSize, BorderStyle, CharCase, Color, ComponentIndex, Ctl3D, Cursor, DragCursor, DragMode, Enabled, Font, Height, HelpContext, HelpKeyword, HelpTypeHideSelection, Hint, Left, MaxLength, Modified, Name, OEMConvert, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PassWordChar, PopupMenu, ReadOnly, SelLengh, SelStart, SelText, ShowHint, Showing, TabOrder, TabStop, Tag, Text, Top, Visible e Width

Principais Métodos

BeginDrag, BringToFront, Clear, ClearSelection, ClientToScreen, CopyToClipBoard, CutToClipBoard, Dragging, EndDrag, FindComponent, Free, GetSelText, GetTextBuf, GetTextLen, Hide, Invalidate,

PasteFromClipBoard, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SelectAll, SendToBack, SetBounds, SetFocus, SetSelTextBuf, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp

TField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena qualquer tipo de dado.

Unit

Na VCL e na CLX:

DB

Principais Propriedades

Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsSQLTimeStamp, AsString, Calculated, CanModify, DataSet, DataSize, DataType, DisplayLabel, DisplayName, DisplayText, DisplayWidth, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, Name, Owner, ReadOnly, Required, Size, Tag, Text, e Visible

Principais Métodos

Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData

Principais Eventos

OnChange, OnGetText, OnSetText e OnValidate

TFieldDef

Descrição

Esse objeto representa um campo de um registro de uma tabela de um banco de dados que armazena qualquer tipo de dado.

Unit

Na VCL e na CLX:

DB

Principais Propriedades

DataType, FieldClass, FieldNo, Name, Required e Size

Principais Métodos

ClassName, ClassParent, ClassType, Create, CreateField, Destroy e Field

Principais Eventos

Esse objeto não possui eventos associados.

TFileListBox

Descrição

O controle TFileListBox é uma caixa de listagem especial que exibe os arquivos do diretório corrente.

Unit

Na VCL:

FileCtrl

Principais Propriedades

Align, BoundsRect, Canvas, ComponentIndex, Color, Controls, Ctrl3D, Cursor, Directory, DragCursor, DragMode, Enabled, FileEdit, FileName, FileType, Font, Handle, Height, HelpContext, HelpKeyword,

HelpTypeHint, IntegralHeight, ItemHeight, ItemIndex, Items, Left, Mask, MultiSelect, Name, Owner, Parent, ParentColor, ParentCtrl3D, ParentFont, ParentShowHint, PopupMenu, Selected, ShowGlyphs, ShowHint, Showing, TabOrder, TabStop, Tag, Top, TopIndex, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, Clear, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Invalidate, ItemAtPos, ItemRect, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp

TFILTERCOMBOBOX

Descrição

O controle TFilterComboBox é uma caixa combo especial que permite ao usuário selecionar filtros de arquivos.

Unit

Na VCL:

FileCtrls

Principais Propriedades

Align, BoundsRect, Color, ComponentIndex, Ctrl3D, Cursor, DragCursor, DragMode, Enabled, FileList, Filter, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, ItemIndex, Items, Left, Mask, Name, Owner, Parent, ParentColor, ParentCtrl3D, ParentFont, SelLength, SelStart, Showing, TabOrder, TabStop, Text, Tag, Top, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, Clear, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SelectAll, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnDropDown, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp

TFINDDIALOG

Descrição

O controle TFindDialog fornece uma caixa de diálogo para pesquisa de texto.

Unit

Na VCL:

Dialogs

Na CLX:

QDialogs

Principais Propriedades

ComponentIndex, Ctl3D, FindText, Handle, HelpContext, HelpKeyword, HelpTypeName, Options, Owner, Position e Tag

Principais Métodos

CloseDialog e Execute

Principais Eventos

OnFind

TFloatField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um número no formato de ponto flutuante.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsString, Calculated, CanModify, Currency,
DataSet, DataSize, DataType, DisplayFormat, DisplayLabel, DisplayName, DisplayText, DisplayWisth,
EditFormat, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, MaxValue,
MinValue, Name, Owner, Precision, ReadOnly, Required, Size, Tag, Text, Value e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate
```

TFont

Descrição

Esse objeto armazena um tipo de fonte.

Unit

Na VCL:

```
Graphics
```

Na CLX:

```
QGraphics
```

Principais Propriedades

```
Color, Handle, Height, Name, Pitch, PixelsPerInch, Size e Style
```

Principais Métodos

```
Assign, Create, Destroy e Free
```

Principais Eventos

```
OnChange
```

TFontDialog

Descrição

O controle TFontDialog fornece uma caixa de diálogo para a seleção de fontes.

Unit

```
Na VCL:
Dialogs
```

Na CLX:

```
QDialogs
```

Principais Propriedades

```
ComponentIndex, Ctl3D, Device, Font, HelpContext, HelpKeyword, HelpTypeMaxFontSize, MinFontSize,
Name, Options, Owner e Tag
```

Principais Métodos

```
Execute
```

Principais Eventos

OnApply

TForm

Descrição

O TForm é o principal componente de uma aplicação desenvolvida em Delphi, pois é nele que são inseridos os demais controles. Um controle do tipo TForm pode ser usado como uma janela, uma caixa de diálogo ou qualquer tipo de formulário para entrada de dados.

Unit

Na VCL:

Forms

Na CLX:

QForms

Principais Propriedades

Active, ActiveControl, ActiveMDIChild, Align, AutoScroll, BorderIcons, BorderStyle, Brush, Caption, Canvas, ClientHandle, ClientHeight, ClientOrigin, ClientRect, ClientWidth, Color, ComponentCount, ComponentIndex, Components, Controls, Ctrl3D, Cursor, DockClientCount, DockClients, Enabled, Font, FormStyle, Handle, Height, HelpContext, HelpFile, HelpKeyword, HelpTypeHint, HorzScrollBar, Icon, KeyPreview, Left, MDIChildCount, MDIChildren, Menu, ModalResult, Name, Owner, Parent, PixelsPerInch, PopupMenu, Position, PrintScale, Scaled, ShowHint, TabOrder, TabStop, Tag, TileMode, Top, VertScrollBar, Visible, Width, WindowMenu e WindowState

Principais Métodos

ArrangeIcons, BringToFront, CanFocus, Cascade, ClientToScreen, Close, CloseQuery, ContainsControl, Create, CreateNew, Destroy, Dragging, FindComponent, Focused, Free, GetFormImage, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl, Invalidate, Next, Previous, Print, Refresh, Release, RemoveComponent, Repaint, ScaleBy, ScreenToClient, ScrollBy, ScrolllnView, SendToBack, SetBounds, SetFocus, SetTextBuf, Show, ShowModal, Tile e Update

Principais Eventos

OnActivate, OnClick, OnClose, OnCloseQuery, OnCreate, OnDestroy, OnDblClick, OnDeactivate, OnDragDrop, OnDragOver, OnEnter, OnExit, OnKeyDown, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp, OnPaint, OnResize e OnShow

TGraphic

Descrição

Esse objeto representa a classe da qual são derivados os objetos TBitmap, TIcon e

TMetafile.

Unit

Na VCL:

Graphics

Na CLX:

QGraphics

Principais Propriedades

Height, Empty e Width

Principais Métodos

ClassName, ClassParent, ClassType, Create, Destroy, Free, LoadFromFile e SaveToFile

Principais Eventos

OnChange

TGraphicField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena uma informação gráfica em um conjunto arbitrário de bytes cujo tamanho não é predefinido.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsString, Calculated, CanModify, DataSet, DataSize,
DataType, DisplayLabel, DisplayName, DisplayText, DisplayWisth, EditMask, EditMaskPtr, FieldName,
FieldNo, Index, IsIndexField, IsNull, Name, Owner, ReadOnly, Required, Size, Tag, Text e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar, LoadFromFile, LoadFromStream,
SaveToFile, SaveToStream e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate
```

TGroupBox

Descrição

O controle TGroupBox permite agrupar componentes que se relacionam, como botões de rádio, por exemplo.

Unit

Na VCL:

```
StdCtrls
```

Na CLX:

```
QStdCtrls
```

Principais Propriedades

```
Align, Caption, Color, Controls, Ctl3D, Cursor, DragCursor, DragMode, Enabled, Font, Height,
HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Owner, Parent, ParentColor, ParentCtl3D,
ParentFont, PopupMenu, Showing, TabOrder, TabStop, Tag, Top, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, ClientToScreen, ContainsControl, Dragging, EndDrag, Focused,
GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy,
SendToBack, SetBounds, SetFocus e SetTextBuf
```

Principais Eventos

```
OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnMouseDown, OnMouseMove e
OnMouseUp
```

THeader

Descrição

O controle THeader é um controle que exibe texto em seções redimensionáveis com o mouse.

Unit

Na VCL:

```
ExtCtrls
```

Na CLX:

```
QExtCtrls
```

Principais Propriedades

```
Align, AllowResize, BoundsRect, BorderStyle, ComponentIndex, Cursor, Enabled, Font, Height,
HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Owner, Parent, ParentFont, ParentShowHint,
Sections, SectionWidth, ShowHint, Showing, TabOrder, TabStop, Tag, Top, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf,
GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack,
SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnEnter, OnExit, OnSized e OnSizing
```

THEADERCONTROL

Descrição

Esse controle é semelhante ao controle THeader e exibe texto em seções redimensionáveis com o mouse.

Unit

Na VCL:

```
ComCtrls
```

Na CLX:

```
QComCtrls
```

Principais Propriedades

```
Align, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect, ClientWidth, ComponentCount,
ComponentIndex, Components, ControlCount, Controls, Cursor, DragCursor, DragMode, Enabled, Font,
Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Owner, Parent, ParentFont,
ParentShowHint, PopupMenu, Sections, ShowHint, Showing, Tag, Top, Visible e Width
```

Principais Métodos

```
Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen,
ContainsControl, ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindComponent, Focused,
Free, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl,
Invalidate, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, ScrollBy,
SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnDragDrop, OnDragOver, OnEndDrag, OnMouseDown, OnMouseMove, OnMouseUp, OnDrawSection, OnResize,
OnSectionClick, OnSectionResize, OnSectionTrack e OnStartDrag
```

THOTKEY

Descrição

O controle THotKey é usado para criar teclas aceleradoras durante a execução do aplicativo. Normalmente está associado a outro controle, ao qual se refere a tecla aceleradora.

Unit

Na VCL:

```
ComCtrls
```

Principais Propriedades

```
Align, AutoSize, BoundsRect, Brush, ClientHeight, ClientOrigin, ClientRect, ClientWidth, ComponentCount,
ComponentIndex, Components, ControlCount, Controls, Cursor, Enabled, Handle, Height, HelpContext,
HelpKeyword, HelpTypeHotKey, InvalidKeys, Left, Modifiers, Name, Owner, Parent, ShowHint, Showing,
TabOrder, TabStop, Tag, Top, Visible e Width
```

Principais Métodos

Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen, ContainsControl, ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindComponent, Focused, Free, GetAssigned, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl, Invalidate, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnEnter, OnExit, OnMouseDown, OnMouseMove e OnMouseUp

TIBCLIENTDATASET

Descrição

Este componente pode ser considerado como uma reunião de três componentes: um IBDataSet, um DatasetProvider e um ClientDataset.

Unit

Na VCL:

IBCuistomDataset

Principais Propriedades

Active, ActiveAggs, AggFields, Aggregates, AggregatesActive, Appserver, AutoCalcFields, BlockReadSize, Bof, Bookmark, CanModify, ChangeCount, CloneSource, CommandText, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Constraints, Data, DataSetField, DataSize, DataSource, DBConnection, DBTransaction, DefaultFields, Delta, Designer, DesignInfo, DisableStringTrim, Eof, FetchOnDemand, FieldCound, FieldDefList, FieldDefs, FieldList, Fields, FieldValues, FileName, Filter, Filtered, FilterOptions, Found, GroupingLevel, HasAppServer, IndexDefs, IndexFieldCount, IndexFieldNames, IndexFields, IndexName, IsUnidirecional, KeyExclusive, KeyFieldCount, KeySize, LogChanges, MasterFields, MasterSource, Modified, Name, ObjectView, Options, Owner, PacketRecords, Params, ReadOnly, RecNo, RecordCount, RecordSize, SparseArrays, State, StatusFilter, Tag, UpdateMode, VCLComObject e XMLData.

Principais Métodos

ActiveBuffer, AddIndex, AfterConstruction, Append, AppendData, AppendRecord, ApplyRange, ApplyUpdates, Assign, BookmarkValid, Cancel, CancelRange, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, ClenupInstance, ClearFields, CloneCursor, Close, CompareBookmarks, ConstraintsDisabled, ControlsDisabled, Create, CreateBlobStream, CreateDataSet, CursorPosChanged, DataRequest, DefaultHandler, Delete, DeleteIndex, Destroy, DestroyComponents, Destroying, DisableConstraints, DisableControls, Dispatch, Edit, EditKey, EditRangeEnd, EditRangeStar, EmptyDataSet, EnableConstraints, EnableControls, Execute, ExecuteAction, FetchBlobs, FetchDetails, FetchParams, FieldAddress, FieldByName, FindComponent, FindField, FindFirst, FindKey, FindLast, FindNearest, FindNext, FindPrior, First, Free, FreeBookmarl, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDatasets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetGroupState, GetIndexInfo, GetIndexNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetNextPacket, GetOptionalParam, GetParentComponent, GetQuoteChar, GotoBookmark, GotoCurrent, GotoKey, GotoNearest, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsImplementorOf, IsLinkedTo, IsSequenced, Last, LoadFromFile, LoadFromStream, Locate, Lookup, MergeChangeLog, MethodAddress, MethodName, MoveBy, NewInstance, Next, Open, Post, Prior, Reconcile, ReferenceInterface, Refresh, RefreshRecord, RemoveComponent, RemoveFreeNotification, Resync, RevertRecord, SaveCallException, SaveToFile, SaveToStream, SetAltRecBuffers, SetFields, SetKey, SetOptionalParam, SetProvider, SetRange, SetRangeEnd, SetRangeStart, SetSubCompnent, Translate, UndoLastChange, UpdateAction, pdateCursorPos, UpdateRecord e UpdateStatus

Principais Eventos

AfterApplyUpdates, AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterExecute, AfterGetParams, AfterGetRecords, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterRowRequest, AfterScroll, AfterUpdateRecord, BeforeApplyUpdates, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeExecute, BeforeGetParams, BeforeGetRecords, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeRowRequest, BeforeScroll, BeforeUpdateRecord, OnCalcFields, OnDeleteError, OnEditError, OnFilterRecord, OnGetTableName, OnNewRecord, OnPostError, OnReconcileError, OnUpdateData e OnUpdateError

TIBDatabase

Descrição

Este componente encapsula uma conexão a um banco de dados do Interbase, através do Mecanismo Interbase Express

Unit

Na VCL:

```
IBDatabase
```

Principais Propriedades

```
ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Connected,
DatabaseName, DatasetCount, DataSets, DBParamByDPB, DBSQLDialect, DefaultTransaction, DesignInfo,
Handle, HandleIsShared, IdleTimer, IsReadOnly, LoginPrompt, Name, Owner, Params, SQLDialect,
SQLObjectCount, SQLObjects, Tag, Temporary, TraceFlags, TransactionCount, Transactions, VCLComObject
```

Principais Métodos

```
AfterConstruction, ApplyUpdates, Assign, BeforeDestruction, ClassInfo, ClassName, ClassNameIs,
ClassParent, ClassType, CleanupInstance, Close, CloseDataSets, Commit, Create, DefaultHandler,
Destroy, DestroyComponents, Destroying, Dispatch, Execute, ExecuteAction, FieldAddress,
FindComponent, FlushSchemaCache, Free, FreeInstance, FreeNotification, FreeOnRelease, GetInterface,
GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, HasParent, InheritsFrom,
InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName, NewInstance, Open,
RemoveComponent, Rollback, SafeCallException, StartTransaction, UpdateAction, ValidateName
```

Principais Eventos

```
AfterConnect, AfterDisconnect, BeforeConnect, BeforeDisconnect, OnDialectDowngradeWarnin,
OnIdleTimer, OnLogin
```

TIBDatabaseInfo

Descrição

Este componente Fornece Informações sobre um banco de dados do Interbase, através do Mecanismo Interbase Express

Unit

Na VCL:

```
IBDatabaseInfo
```

Principais Propriedades

```
Allocation, BackoutCount, BaseLevel, ComObject, ComponentCount, ComponentIndex, Components,
ComponentState, ComponentStyle, CurrentMemory, Database, DBFileName, DBImplementationClass,
DBImplementationNo, DBSiteName, DBSQLDialect, DeleteCount, DesignInfo, ExpungeCount, Fetches,
ForcedWrites, InsertCount, Marks, MaxMemory, Name, NoReserve, NumBuffers, ODSMajorVersion,
ODSMinorVersion, Owner, PageSize, PurgeCount, ReadIdxCount, ReadOnly, Reads, ReadSeqCount,
SweepInterval, Tag, UpdateCount, UserNames, VCLComObject, Version, Writes
```

Principais Métodos

```
AfterConstruction, Assign, BeforeDestruction, Call, ClassInfo, ClassName, ClassNameIs, ClassParent,
ClassType, CleanupInstance, Create, DefaultHandler, Destroy, DestroyComponents, Destroying,
Dispatch, ExecuteAction, FieldAddress, FindComponent, Free, FreeInstance, FreeNotification,
FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent,
HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName,
NewInstance, RemoveComponent, SafeCallException, UpdateAction
```

Principais Eventos

Este componente não possui eventos associados.

TIBDataset

Descrição

Este componente Permite que se executem comandos sql diretamente em um banco de dados do Interbase, através do Mecanismo Interbase Express.

Unit

Na VCL:

```
IBCustomDataset
```

Principais Propriedades

```
Active, AggFields, AutoCalcFields, Bof, Bookmark, BufferChunks, CachedUpdates, ComObject,
ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Database, DatasetField,
DataSource, DBHandle, DefaultFields, DeleteSQL, Designer, DesignInfo, Eof, FieldCount, FieldDefList,
FieldDefs, FieldList, Fields, FieldValues, Found, InsertSQL, Modified, ModifySQL, Name, ObjectView,
Owner, Params, Prepared, Qdelete, Qinsert, Qmodify, Qrefresh, Qselect, RecordCount, RefreshSQL,
SelectSQL, SparseArrays, State, StatementType, Tag, Transaction, TRHandle, UpdateObject,
UpdateRecordTypes, UpdatesPending, VCLComObject
```

Principais Métodos

```
ActiveBuffer, AfterConstruction, Append, AppendRecord, ApplyUpdates, Assign, BeforeDestruction,
CachedUpdateStatus, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent,
ClassType, CleanupInstance, ClearFields, Close, CompareBookmarks, ControlsDisabled, Create,
CreateBlobStream, CursorPosChanged, DefaultHandler, Delete, Destroy, DestroyComponents, Destroying,
DisableControls, Dispatch, Edit, EnableControls, ExecuteAction, FetchAll, FieldAddress, FieldByName,
FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark,
FreeInstance, FreeNotification, FreeOnRelease, GetBookmark, GetCurrentRecord, GetDetailDataSets,
GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetInterface, GetInterfaceEntry,
GetInterfaceTable, GetNamePath, GetParentComponent, GetProviderAttributes, GotoBookmark, HasParent,
InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsLinkedTo,
Last, Locate, LocateNext, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, Open, Post,
Prepare, Prior, RecordModified, Refresh, RemoveComponent, Resync, RevertRecord, SafeCallException,
SetFields, Translate, Undelete, UnPrepare, UpdateAction, UpdateCursorPos, UpdateRecord, UpdateStatus
```

Principais Eventos

```
AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh,
AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost,
BeforeRefresh, BeforeScroll, DatabaseDisconnected, DatabaseDisconnecting, DatabaseFree, OnCalcFields,
OnDeleteError, OnEditError, OnEditError, OnFilterRecord, OnPostError, OnUpdateError, OnUpdateRecord,
```

TIBEvents

Descrição

Este componente Permite a uma aplicação responder a eventos enviados pelo servidor de banco de dados do Interbase, através do Mecanismo Interbase Express.

Unit

Na VCL:

```
IBEvents
```

Principais Propriedades

```
ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Databases,
DesignInfo, Events, Name, Owner, Queued, Registered, Tag, VCLComObject
```

Principais Métodos

```
AfterConstruction, Assign, BeforeDestruction, CancelEvents, ClassInfo, ClassName, ClassNameIs,
ClassParent, ClassType, CleanupInstance, Create, DefaultHandlerm Destroy, DestroyComponents, Destroying,
Dispatch, ExecuteAction, FieldAddress FindComponent, Free, FreeInstance, FreeNotification, FreeOnRelease,
GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, HasParent,
InheritsFrom, InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName, NewInstance,
QueEvents, RegisterEvents, RemoveComponent, SafeCallException, UnregisterEvents, UpdateAction
```

Principais Eventos

OnEventAlert

TIBQuery

Descrição

Este componente permite acesso via declarações SQL a tabelas de bancos de dados do interbase através do Mecanismo Interbase Express.

Unit

Na VCL:

IBQuery

Principais Propriedades

Active, AggFields, AutoCalcFields, BlockReadSize, Bof, Bookmark,

CachedUpdates, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Constraints, Database, DataSetField, DBHandle, DefaultFields, Designer, DesignInfo, Eof, FieldCount, FieldDefList, FieldDefs, FieldList, Fields, FieldValues, FilterOptions, Found, GenerateParamNames, Modified, Name, ObjectView, Owner, ParamCheck, ParamCount, Params, Prepared, RecNo, RecordCount, RecordSize, RowsAffected, SparseArrays, SQL, State, StmtHandle, TableTypes, Tag, Text, Transaction, TRHandle, Unidirecional, UpdateObject, UpdateRecordTypes, UpdatesPending, VCLComObject

Principais Métodos

ActiveBuffer, AfterConstruction, Append, AppendRecord, ApplyUpdates, Assign, BeforeDestruction, BookmarkValid, Cancel, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CursorPosChanged, DefaultHandler, Delete, Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EnableControls, ExecSQL, ExecuteAction, FetchAll, FieldAddress, FieldByName, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GetProviderAttributes, GotoBookmark, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsLinkedTo, IsSequenced, Last, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, Open, ParamByName, Post, Prepare, Prior, Refresh, RemoveComponent, Resync, RevertRecord, SafeCallException, SetFields, Translate, Unprepare, UpdateAction, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnNewRecord, OnPostError, OnUpdateError, OnUpdateRecord

TIBSQL

Descrição

Este componente Permite que se executem comandos sql Unidirecionais sem overhead diretamente em um banco de dados do Interbase, através do Mecanismo Interbase Express.

Unit

Na VCL:

IBSQL

Principais Propriedades

Bof, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Database, DBHandle, DesignInfo, Eof, FieldIndex, Fields, GenerateParamNames, GoToFirstRecordOnExecute, Handle, Name, Open, Owner, ParamCheck, Params, Plan, Prepared, RecordCount, RowsAffected, SQL, SQLType, Tag, Transaction, TRHandle, UniqueRelationName, VCLComObject

Principais Métodos

AfterConstruction, Assign, BatchInput, BatchOutput, BeforeDestruction, Call, CheckClosed, CheckOpen, CheckValidStatement, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, Close, Create, Current, DefaultHandler Destroym, DestroyComponents, Destroying, Dispatch, ExecQuery, ExecuteAction, FieldAddress, FieldByName, FindComponent, Free, FreeHandle, FreeInstance, FreeNotification, FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName, NewInstance, Next, Prepare, RemoveComponent, SafeCallException, UpdateAction

Principais Eventos

OnSQLChanging

TIBSQLMonitor

Descrição

Este componente Permite o monitoramento de comandos SQl enviados para serem executados no servidor de um banco de dados do Interbase, através do Mecanismo Interbase Express.

Unit

Na VCL:

IBSQLMonitor

Principais Propriedades

ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DesignInfo, Name, Owner, Tag, VCLComObject

Principais Métodos

AfterConstruction, Assign, BeforeDestruction, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, Create, DefaultHandlerm Destroy, DestroyComponents, Destroying, Dispatch, ExecuteAction, FieldAddress FindComponent, Free, FreeInstance, FreeNotification, FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName, NewInstance, RemoveComponent, SafeCallException, UpdateAction

Principais Eventos

OnSQL

TIBStoredProc

Descrição

Este componente permite acesso via declarações SQL a tabelas de bancos de dados do interbase através do Mecanismo Interbase Express.

Unit

Na VCL:

IBStoredProc

Principais Propriedades

Active, AggFields, AutoCalcFields, BlockReadSize, Bof, Bookmark,

CachedUpdates, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Constraints, Database, DataSetField, DBHandle, DefaultFields, Designer, DesignInfo, Eof, FieldCount, FieldDefList, FieldDefs, FieldList, Fields, FieldValues, FilterOptions, Found, Modified, Name, NameList, ObjectView, Owner, ParamCount, Params, Prepared, RecNo, RecordCount, RecordSize, SparseArrays, State, StmtHandle, StoredProcName, TableTypes, Tag, Transaction, TRHandle, UpdateRecordTypes, UpdatesPending, VCLComObject

Principais Métodos

ActiveBuffer, AfterConstruction, Append, AppendRecord, ApplyUpdates, Assign, BeforeDestruction, BookmarkValid, Cancel, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Close, CompareBookmarks, ControlsDisabled,

Create, CreateBlobStream, CursorPosChanged, DefaultHandler, Delete, Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EnableControls, ExecProc, ExecuteAction, FetchAll, FieldAddress, FieldByName, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GetProviderAttributes, GotoBookmark, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsLinkedTo, IsSequenced, Last, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, Open, ParamByName, Post, Prepare, Prior, Refresh, RemoveComponent, Resync, RevertRecord, SafeCallException, SetFields, Translate, Unprepare, UpdateAction, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnNewRecord, OnPostError, OnUpdateError, OnUpdateRecord

TIBTABLE

Descrição

Este componente permite acesso direto a tabelas de bancos de dados através do Mecanismo Interbase Express.

Unit

Na VCL:

IBTable

Principais Propriedades

Active, AggFields, AutoCalcFields, BlockReadSize, Bof, Bookmark, BufferChunks, CachedUpdates, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Constraints, CurrentDBKey, Database, DataSetField, DBHandle, DefaultFields, DefaultIndex, Designer, DesignInfo, Eof, Exists, FieldCount, FieldDefList, FieldDefs, FieldList, Fields, FieldValues, Filter, Filtered, FilterOptions, Found, IndexDefs, IndexFieldCount, IndexFieldNames, IndexFields, IndexName, MasterFields, MasterSource, Modified, Name, ObjectView, Owner, ReadOnly, RecNo, RecordCount, RecordSize, SparseArrays, State, StoreDefs, TableName, TableNames, TableTypes, Tag, Transaction, TRHandle, UniDirectional, UpdateObject, UpdateRecordTypes, UpdatesPending, VCLComObject

Principais Métodos

ActiveBuffer, AddIndex, AfterConstruction, Append, AppendRecord, ApplyUpdates, Assign, BeforeDestruction, BookmarkValid, Cancel, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CreateTable, CursorPosChanged, DefaultHandler, Delete, DeleteIndex, DeleteTable, Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EmptyTable, EnableControls, ExecuteAction, FetchAll, FieldAddress, FieldByName, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetIndexNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GetProviderAttributes, GotoBookmark, GotoCurrent, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsLinkedTo, IsSequenced, Last, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, Open, Post, Prior, Refresh, RemoveComponent, Resync, RevertRecord, SafeCallException, SetFields, Translate, UpdateAction, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnNewRecord, OnPostError, OnUpdateError, OnUpdateRecord

T IBTransaction

Descrição

Este componente fornece controle discreto de transações para uma ou mais conexões a um banco de dados do Interbase, através do Mecanismo Interbase Express

Unit

Na VCL:

```
IBDatabase
```

Principais Propriedades

```
Active, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle,
DatabaseCount, Databases, DefaultAction, DefaultDatabase, DesignInfo, Handle, HandleIsShared,
IdleTimer, InTransaction, Name, Owner, Params, SQLObjectCount, SQLObjects, Tag, TPB, TPBLength,
VCLComObject
```

Principais Métodos

```
AddDatabase, AfterConstruction, Assign, BeforeDestruction, Call, CheckDatabasesInList, CheckInTransaction,
CheckNotInTransaction, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, Commit,
CommitRetaining, Create, DefaultHandler, Destroy, DestroyComponents, Destroying, Dispatch, ExecuteAction,
FieldAddress, FindComponent, FindDatabase, Free, FreeInstance, FreeNotification, FreeOnRelease,
GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, HasParent,
InheritsFrom, InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName, NewInstance,
RemoveComponent, RemoveDatabase, RemoveDatabases, Rollback, RollbackRetaining, SafeCallException,
StartTransaction, UpdateAction
```

Principais Eventos

```
OnIdleTimer
```

TIBUpdateSQL

Descrição

O componente TUpdateSQL fornece um meio alternativo de se atualizar bancos de dados, através do mecanismo Interbase Express, armazenando um comando para cada tipo de declaração SQL.

Unit

Na VCL:

```
IBUPdateSQL
```

Principais Propriedades

```
ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DataSet,
DeleteSQL, DesignInfo, InsertSQL, ModifySQL, Name, Owner, Query, RefreshSQL, SQL, Tag, VCLComObject
```

Principais Métodos

```
AfterConstruction, Apply, Assign, BeforeDestruction, ClassInfo, ClassName, ClassNameIs, ClassParent,
ClassType, CleanupInstance, Create, DefaultHandler, Destroy, DestroyComponents, Destroying,
Dispatch, ExecSQL, ExecuteAction, FieldAddress, FindComponent, Free, FreeInstance, FreeNotification,
FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent,
HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName,
NewInstance, RemoveComponent, SafeCallException, SetParams, UpdateAction
```

Principais Eventos

Esse componente não possui métodos associados.

TIcon

Descrição

Esse objeto representa um ícone (arquivo no formato *.ICO).

Unit

Na VCL:

```
Graphics
```

Na CLX:

```
QGraphics
```

Principais Propriedades

```
Handle, Height, Empty e Width
```

Principais Métodos

```
Assign, ClassName, ClassParent, ClassType, Create, Destroy, Free, LoadFromFile e
SaveToFile
```

Principais Eventos

```
OnChange
```

TImage

Descrição

Esse componente exibe uma imagem em um formulário.

Unit

Na VCL:

```
StdCtrls
```

Na CLX:

```
QStdCtrls
```

Principais Propriedades

```
Align, AutoSize, BoundsRect, BorderStyle, Center, ComponentIndex, Cursor, DragCursor, DragMode,
Enabled, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Owner, Parent, ParentShowHint,
Picture, PopupMenu, ShowHint, Showing, Stretch, TabOrder, TabStop, Tag, Top, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, ClientToScreen, Dragging, EndDrag, Focused, Hide, Invalidate, Refresh,
Repaint, ScaleBy, ScreenToClient, SendToBack, SetBounds, Show e Update
```

Principais Eventos

```
OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress,
OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp
```

TImageList

Descrição

Esse controle é usado como um contêiner para imagens gráficas, sendo capaz de armazenar um grande número de imagens de mesmo tamanho.

Unit

Na VCL:

```
Controls
```

Na CLX:

```
QImgList
```

Principais Propriedades

```
AllocBy, BkColor, BlendColor, ComponentIndex, Count, DragCursor, Dragging, DrawingStyle, Enabled,
Handle, Height, ImageType, Masked, Name, Owner, ShareImages, Tag e Width
```

Principais Métodos

Add, AddIcon, AddMasked, Assign, BeginDrag, Clear, Create, CreateSize, Delete, Destroy, DragLock, DragMode, DragUnLock, Draw, DrawOverlay, EndDrag, FileLoad, Free, GetBitmap, GetHotSpot, GetIcon, GetImageBitmap, GetMaskBitmap, GetResource, HandleAllocated, HideDragImage, Insert, InsertIcon, InsertMasked, Move, Overlay, RegisterChanges, Replace, ReplaceIcon, ReplaceMasked, ResourceLoad, SetDragImage, ShowDragImage e UnRegisterChanges

Principais Eventos

OnChange

TIniFile

Descrição

Esse objeto permite que sua aplicação realize operações de leitura e gravação em arquivos com extensão .INI.

Unit

Na VCL e na CLX:

IniFiles

Principais Propriedades

Esse objeto não possui propriedades associadas.

Principais Métodos

ClassName, ClassParent, ClassType, Create, Destroy, EraseSection, FileName, Free, ReadBool, ReadInteger, ReadSection, ReadSectionValues, ReadString, WriteBool, WriteInteger e WriteString

Principais Eventos

Esse objeto não possui eventos associados.

TIntegerField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um número no formato inteiro, variando de -2,147,483,648 a 2,147,483,647.

Unit

Na VCL e na CLX:

DB

Principais Propriedades

Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsSQLTimeStamp, AsString, Calculated, CanModify, DataSet, DataSize, DataType, DisplayFormat, DisplayLabel, DisplayName, DisplayText, DisplayWisth, EditFormat, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, MaxValue, MinValue, Name, Owner, Precision, ReadOnly, Required, Size, Tag, Text, Value e Visible

Principais Métodos

Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData

Principais Eventos

OnChange, OnGetText, OnSetText e OnValidate

THTTPFile

Unit

Na VCL e na CLX:

IWFileParser

Descrição

Realiza o Parser para a geração da resposta HTML.

Propriedades

Esta classe não possui propriedades associadas.

Métodos

Create, Destroy

Eventos

Esta classe não possui eventos associados.

TIWAppForm

Unit

Na VCL e na CLX:

IWAppForm

Descrição

Esta classe representa um formulário em uma aplicação Intraweb.

Propriedades

ActiveControl, Background, BackgroundColor, DataSetList, ExtraHeader, FormAction, HandleTabs, HiddenParams, JavaScript, JavaScriptOnce, LinkColor, Params, Released, ScriptFiles, ShowHint, StyleSheet, SupportedBrowsers, TabOrderList, TemplateProcessor, TextColor, Title, VLinkColor, WebApplication

Métodos

AddScriptFile, AddToControlList, AddToInitProc, AddToIWCLInitProc, AddValidation, CacheImage, Create, CreateNew, Destroy, DoDefaultAction, DoPreview, ExecuteForm, FixupTabList, GenerateForm, Hide, Release, RenderControl, Show

Eventos

OnAfterRender, OnCreate, OnDefaultAction, OnDestroy, OnRender

TIWApplet

Unit

Na VCL e na CLX:

IWApplet

Descrição

Este componente permite executar uma applet em uma aplicação Intraweb.

Propriedades

Align, Alignment, AltText, Anchors, AppletName, Archive, Canvas, Caption, ClassFile, CodeBase, Clip, Color, ControlEncode, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HorizSpace, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, Params, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, VertSpace, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag

TIWApplication

Unit

Na VCL e na CLX:

IWApplication

Descrição

Esta classe representa uma aplicação Intraweb.

Propriedades

ActiveForm, AppID, AuthUser, Browser, Data, FileList, FormAction, FormCount, Forms, IP, LastAccess, ReferringURL, Request, Response, RunParams, Terminated, TerminateMessage, TerminateURL, TrackID, URLBase

Métodos

AddDataModule, Create, Destroy, ExecuteActiveForm, GenerateActiveForm, Initialize, MarkAccess, RemoveDataModule, RemoveForm, SendFile, SendStream, SetActiveForm, ShowMessage, Terminate, TerminateAndRedirect

Eventos

TIWButton

Unit

Na VCL e na CLX:

IWCompButton

Descrição

Versão Intraweb do componente Button.

Propriedades

Align, Anchors, ButtonType, Canvas, Caption, Clip, Color, Confirmation, ControlEncode, DesignMode, DoSubmitValidation, Enabled, ExtraTagParams, Font, Form, FriendlyName, HotKey, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, TabOrder, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnClick, OnHTMLTag

TIWCheckBox

Unit

Na VCL e na CLX:

IWCompCheckBox

Descrição

Versão Intraweb do componente CheckBox.

Propriedades

Align, Anchors, Canvas, Caption, Checked, Clip, Color, ControlEncode, DesignMode, DoSubmitValidation, Editable, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, Style, SupportedScriptEvents, TabOrder, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnClick, OnHTMLTag

TIWCOMBOBOX

Unit

Na VCL e na CLX:

IWCompListbox.

Descrição

Versão Intraweb do componente ComboBox.

Propriedades

Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, ItemIndex, Items, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, Selected, ShowHint, Sorted, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, ResetSelection, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnChange, OnHTMLTag

TIWCONTROL

Unit

Na VCL e na CLX:

Descrição

Esta é a classe-base da maioria dos controles e componentes Intraweb.

Propriedades

Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag

TIWDBCHECKBOX

Unit

Na VCL e na CLX:

IWDBStdCtrls

Descrição

Versão Intraweb do componente DBCheckBox.

Propriedades

Align, Anchors, Canvas, Caption, Checked, Clip, Color, ControlEncode, DataField, Datasource,DesignMode, DoSubmitValidation, Editable, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, Style, SupportedScriptEvents, TabOrder, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnClick, OnHTMLTag

TIWDBComboBox

Unit

Na VCL e na CLX:

IWDBStdCtrls

Descrição

Versão Intraweb do componente DBComboBox.

Propriedades

Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, DataField, Datasource, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, ItemIndex, Items, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, Selected, ShowHint, Sorted, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, ResetSelection, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnChange, OnHTMLTag

TIWDBEdit

Unit

Na VCL e na CLX:

IWDBStdCtrls.

Descrição

Versão Intraweb do componente DBEdit.

Propriedades

Align, Anchors, BGColor, Canvas, Caption, Clip, Color, ControlEncode, DataField, Datasource, DesignMode, DoSubmitValidation, Editable, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, MaxLength, PasswordPrompt, ParentShowHint, ReadOnly, RenderSize, Required, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag, OnSubmit

TIWDBFile

Unit

Na VCL e na CLX:

IWDBStdCtrls.

Descrição

Permite armazenar um arquivo em um banco de dados.

Propriedades

Align, Anchors, AutoEditable, BGColor, Canvas, Caption, Clip, Color, ContentType, ControlEncode, DataField, Datasource, DesignMode, DoSubmitValidation, Editable, ExtraTagParams, FileData, FileName, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, MaxLength, ParentShowHint, ReadOnly, RenderSize, Required, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, TabOrder, Text, UseFrame, ValueChecked, ValueUnchecked, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SaveToFile, SaveToStream, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag, OnReceivedFile, OnSubmit

TIWDBGRID

Unit

Na VCL e na CLX:

IWDBStdCtrls.

Descrição

Versão Intraweb do componente DBGrid.

Propriedades

Align, Anchors, BGColor, BorderColors, BorderSize, Borderstyle, Canvas, Caption, Cell, CellPadding, CellSpacing, Clip, Color, ColumnCount, Columns, ControlEncode, CurrentField, DataField, Datasource, DesignMode, ExtraTagParams, Font, FooterRowCount, Form, FrameBuffer, FriendlyName, FromStart, HighlightColor, HighlightRows, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, Lines, Options, ParentShowHint, RecordCount, RefreshMode, RenderSize, RollOver, RollOverColor, RowAlternateColor, RowClick, RowCount, RowCurrentColor, RowHeaderColor, RowIsCurrent, RowLimit, ScriptEvents, ScriptFiles, ShowHint, Summary, SupportedScriptEvents, Text, UseFrame, UseWidth, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, CellExists, Clear, ColorToRGBString, Create, CreateImplicitColors, DeleteColumn, DeleteRow, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RefreshData, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag, OnRenderCell

TIWDBIMAGE

Unit

Na VCL e na CLX:

IWDBStdCtrls.

Descrição

Versão Intraweb do componente DBImage.

Propriedades

Align, AllText, Anchors, AutoSize, Canvas, Caption, Clip, Color, Confirmation, ControlEncode, Cursosr, DataField, Datasource, DesignMode, DoSubmitValidation, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, JPegOptions, ParentShowHint, Picture, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseBorder, UseFrame, UseSize, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag

TIWDBListBox

Unit

Na VCL e na CLX:

IWDBStdCtrls.

]Descrição

Versão Intraweb do componente DBListBox.

Propriedades

Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, Datafield, Datasource, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, ItemIndex, Items, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, Selected, ShowHint, Sorted, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, ResetSelection, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnChange, OnHTMLTag

TIWDBLookupComboBox

Unit

Na VCL e na CLX:

IWDBStdCtrls.

Descrição

Versão Intraweb do componente DBLookupComboBox.

Propriedades

Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, Datafield, Datasource, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, ItemIndex, Items, IWCLInitProc, JavaScriptOnce, KeyField, ListField, Listsource, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, Selected, ShowHint, Sorted, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, ResetSelection, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnChange, OnHTMLTag

TIWDBLookupListBox

Unit

Na VCL e na CLX:

IWDBStdCtrls.

Descrição

Versão Intraweb do componente DBLookupListBox.

Propriedades

Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, Datafield, Datasource, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, ItemIndex, Items, IWCLInitProc, JavaScriptOnce, KeyField, ListField, Listsource, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, Selected, ShowHint, Sorted, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, ResetSelection, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnChange, OnHTMLTag

TIWDBMEMO

Unit

Na VCL e na CLX:

IWDBStdCtrls.

Descrição

Versão Intraweb do componente DBMemo.

Propriedades

Align, Anchors, AutoEditable, Canvas, Caption, Clip, Color, ControlEncode, DataField, Datasource, DesignMode, Editable, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, Lines, ParentShowHint, RawText, ReadOnly, RenderSize, Required, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, TabOrder, Text, UseFrame, Visible, WantReturns, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag

TIWDBNAVIGATOR

Unit

Na VCL e na CLX:

IWDBStdCtrls.

Descrição

Versão Intraweb do componente DBNavigator.

Propriedades

Align, Anchors, Canvas, Caption, Clip, Color, Confirmations, CustomImages, ControlEncode, Datasource, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, ImageHeight, ImageWidth, InitProcCode, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, Visible, VisibleButtons, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, Submit, SupportsSubmit, ToJPEGFile

Eventos

OnCancel, OnDelete, OnEdit, OnFirst, OnHTMLTag, OnInsert, OnLast, OnNext, OnPost, OnPrior, OnRefresh, Orientation

TIWDBText

Unit

Na VCL e na CLX:

IWDBStdCtrls.

Descrição

Versão Intraweb do componente DBText.

Propriedades

Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, Datafield, Datasource, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, Lines, ParentShowHint, RaeText, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, Visible, WantReturns, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag

TIWEdit

Unit

Na VCL e na CLX:

IWCompEdit

Descrição

Versão Intraweb do componente Edit.

Propriedades

Align, Anchors, BGColor, Canvas, Caption, Clip, Color, ControlEncode, DesignMode, DoSubmitValidation, Editable, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, MaxLength, PasswordPrompt, ParentShowHint, ReadOnly, RenderSize, Required, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag, OnSubmit

TIWFile

Unit

Na VCL e na CLX:

IWCompEdit

Descrição

Exibe o texto de um arquivo.

Propriedades

Align, Anchors, AutoEditable, BGColor, Canvas, Caption, Clip, Color, ContentType, ControlEncode, DesignMode, DoSubmitValidation, Editable, ExtraTagParams, FileData, FileName, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, MaxLength, ParentShowHint, ReadOnly, RenderSize, Required, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, TabOrder, Text, UseFrame, ValueChecked, ValueUnchecked, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SaveToFile, SaveToStream, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag, OnReceivedFile, OnSubmit

TIWFORM

Unit

Na VCL e na CLX:

IWForm.

Descrição

Classe-base de formulários Intraweb.

Propriedades

ActiveControl, Background, BackgroundColor, DataSetList, ExtraHeader, FormAction, HandleTabs, HiddenParams, JavaScript, JavaScriptOnce, LinkColor, Params, ScriptFiles, ShowHint, StyleSheet, SupportedBrowsers, TabOrderList, TemplateProcessor, TextColor, Title, VLinkColor, WebApplication

Métodos

AddScriptFile, AddToControlList, AddToInitProc, AddToIWCLInitProc, AddValidation, CacheImage, Create, CreateNew, Destroy, DoDefaultAction, DoPreview, ExecuteForm, FixupTabList, GenerateForm, RenderControl

Eventos

OnAfterRender, OnCreate, OnDefaultAction, OnDestroy, OnRender

TIWGRID

Unit

Na VCL e na CLX:

IWGrid

Descrição

Versão Intraweb do componente Grid.

Propriedades

Align, Anchors, BGColor, BorderColors, BorderSize, Borderstyle, Canvas, Caption, Cell, CellPadding, CellSpacing, Clip, Color, ColumnCount, ControlEncode, DesignMode, ExtraTagParams, Font, Form, FrameBuffer, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, Lines, ParentShowHint, RenderSize, RowCount, ScriptEvents, ScriptFiles, ShowHint, Summary, SupportedScriptEvents, Text, UseFrame, UseWidth, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, CellExists, Clear, ColorToRGBString, Create, DeleteColumn, DeleteRow, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnCellClick, OnHTMLTag, OnRenderCell

TIWImage

Unit

Na VCL e na CLX:

IWExtCtrls

Descrição

Versão Intraweb do componente Image.

Propriedades

Align, AllText, Anchors, AutoSize, Canvas, Caption, Clip, Color, Confirmation, ControlEncode, Cursosr, DesignMode, DoSubmitValidation, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, JPegOptions, ParentShowHint, Picture, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseBorder, UseFrame, UseSize, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnClick, OnHTMLTag e OnMouseDown

TIWImageFile

Unit

Na VCL e na CLX:

IWExtCtrls

Descrição

Exibe uma imagem proveniente de um arquivo.

Propriedades

Align, AllText, Anchors, AutoSize, Canvas, Caption, Clip, Color, Confirmation, ControlEncode, Cursosr, DesignMode, DoSubmitValidation, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, ImageFile, InitProcCode, IWCLInitProc, JavaScriptOnce, JPegOptions, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseBorder, UseFrame, UseSize, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnClick, OnHTMLTag e OnMouseDown

TIWLabel

Unit

Na VCL e na CLX:

IWCompLabel

Descrição

Versão Intraweb do componente Label.

Propriedades

Align, Anchors, AutoSize, Canvas, Caption, Clip, Color, ControlEncode, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, ParentShowHint, RawText, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag

TIWLINK

Unit

Na VCL e na CLX:

IWHTMLCtrls

Descrição

Representa um link de uma página HTML

Propriedades

Align, Anchors, AutoEditable, Canvas, Caption, Clip, Color, ControlEncode, DesignMode, DoSubmitValidation, Enabled, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, ValueChecked, ValueUnchecked, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HookEvents, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnClick, OnHTMLTag

TIWLIST

Unit

Na VCL e na CLX:

IWHTMLCtrls

Descrição

Representa uma lista de itens de uma página HTML.

Propriedades

Align, Anchors, AutoEditable, Canvas, Caption, Clip, Color, ControlEncode, DataField, Datasource, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, Items, IWCLInitProc, JavaScriptOnce, Numbered, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, ValueChecked, ValueUnchecked, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag

TIWLISTBOX

Unit

Na VCL e na CLX:

IWCompListbox

Descrição

Versão Intraweb do componente ListBox.

Propriedades

Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, ItemIndex, Items, IWCLInitProc, JavaScriptOnce, Multiselect, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, Selected, ShowHint, Sorted, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, ResetSelection, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnChange, OnHTMLTag

TIWMemo

Unit

Na VCL e na CLX:

IWCompMemo

Descrição

Versão Intraweb do componente Memo.

Propriedades

Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, DesignMode, Editable, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, Lines, ParentShowHint, RawText, ReadOnly, RenderSize, Required, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, TabOrder, Text, UseFrame, Visible, WantReturns, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag

TIWRectangle

Unit

Na VCL e na CLX:

IWHTMLCtrls

Descrição

Representa uma figura retangular em uma página HTML.

Propriedades

Align, Alignment, nchors, AutoEditable, Canvas, Caption, Clip, Color, ControlEncode, DataField, Datasource, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, VAlign, ValueChecked, ValueUnchecked, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos
OnHTMLTag

TIWREGION

Unit

Na VCL e na CLX:

IWHTMLCtrls

Descrição

Representa uma região de uma página HTML.

Propriedades
Align, Anchors, Color, Rectangle, TabOrder, Visible

Métodos
Create, Destroy

Eventos

TIWTIMER

Unit

Na VCL e na CLX:

IWExtCtrls

Descrição

Versão Intraweb do componente Timer.

Propriedades
Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, DesignMode, Enabled, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, Interval, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, UseFrame, Visible, WebApplication, ZIndex

Métodos
AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos
OnHTMLTag, OnTimer

TIWTREEVIEW

Unit

Na VCL e na CLX:

IWTreeView

Descrição

Versão Intraweb do componente TreeView.

Propriedades
Align, Anchors, Canvas, Caption, Clip, Color, ControlEncode, DesignMode, ExtraTagParams, Font, Form, FriendlyName, HorScrollbarVisible, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, Items, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, Text, TreeViewImages, UseFrame, VertScrollbarVisible, Visible, WebApplication, ZIndex

Métodos
AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnHTMLTag

TIWURL

Unit

Na VCL e na CLX:

IWHTMLControls

Descrição

Representa um link para uma URL em uma página HTML.

Propriedades

Align, Anchors, AutoEditable, Canvas, Caption, Clip, Color, ControlEncode, DesignMode, DoSubmitValidation, Enabled, ExtraTagParams, Font, Form, FriendlyName, HTML, HTMLLeft, HTMLName, HTMLTop, InitProcCode, IWCLInitProc, JavaScriptOnce, ParentShowHint, RenderSize, ScriptEvents, ScriptFiles, ShowHint, SupportedScriptEvents, TargetOptions, TerminateApp, Text, URL, UseFrame, UseTarget, ValueChecked, ValueUnchecked, Visible, WebApplication, ZIndex

Métodos

AddScriptFile, AddToInitProc, AddToIWCLInitProc, AddToJavaScriptOnce, Clear, ColorToRGBString, Create, Destroy, DoSubmit, HookEvents, HTMLColor, Invalidate, MakeHTMLTag, PaintTo, RenderCSSClass, RenderHTML, RenderScripts, RenderStyle, SetLayoutMgr, SetRenderData, SupportsSubmit, ToJPEGFile

Eventos

OnClick, OnHTMLTag

TLabel

Descrição

O controle TLabel permite a exibição de um texto em um formulário.

Unit

Na VCL:

StdCtrls

Na CLX:

QStdCtrls

Principais Propriedades

Align, Alignment, AutoSize, BoundsRect, Caption, Color, ComponentIndex, Cursor, DragCursor, DragMode, Enabled, FocusControl, Font, Height, Hint, Left, Name, Owner, Parent, ParentColor, ParentFont, ParentShowHint, PopupMenu, ShowAccelChar, ShowHint, Tag, Top, Transparent, Visible, Width e WordWrap

Principais Métodos

BeginDrag, BringToFront, ClientToScreen, Dragging, EndDrag, GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, SendToBack, SetBounds, SetTextBuf, Show e Update

Principais Eventos

OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnMouseDown, OnMouseMove e OnMouseUp

TList

Descrição

Esse objeto é utilizado para manter uma lista de objetos.

Unit

Na VCL e na CLX:

Classes

Principais Propriedades

Capacity, Count, Items e List

Principais Métodos

Add, ClassName, ClassParent, ClassType, Create, Clear, Delete, Destroy, Exchange, Expand, First, Free, IndexOf, Insert, Last, Pack e Remove

Principais Eventos

Esse objeto não possui eventos associados.

TLISTBOX

Descrição

O controle TListBox consiste em uma caixa de listagem na qual o usuário pode selecionar um ou mais itens.

Unit

Na VCL:

StdCtrls

Na CLX:

QStdCtrls

Principais Propriedades

Align, BorderStyle, Canvas, Color, Columns, ComponentIndex, Ctl3D, Cursor, DragCursor, DragMode, Enabled, ExtendedSelect, Fields, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, ItemIndex, IntegralHeight, ItemHeight, Items, Left, MultiSelect, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu, SelCount, Selected, ShowHint, Showing, Sorted, Style, TabOrder, TabStop, Tag, Top, TopIndex, Visible e Width

Principais Métodos

BeginDrag, BringToFront, Clear, ClientToScreen, Dragging, EndDrag, GetTextBuf, GetTextLen, Hide, ItemAtPos, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnClick, OnDblClick, OnDragDrop, OnDragOver, OnDrawItem, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMeasureItem, OnMouseDown, OnMouseMove e OnMouseUp

TLISTVIEW

Descrição

Esse controle permite a exibição de uma lista de itens contendo textos e imagens (ícones).

Unit

Na VCL:

ComCtrls

Na CLX:

QComCtrls

Principais Propriedades

Align, AllocBy, BorderStyle, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect, ClientWidth, Color, Columns, ComponentCount, ComponentIndex, Components, ControlCount, Controls, Ctl3D, Cursor, DragCursor, DragMode, DropTarget, Focused, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeHideSelection, IconOptions, Items, LargeImages, Left, MultiSelect, Name, Owner, Parent, ParentShowHont, PopupMenu, ReadOnly, Selected, ShowColumnHeaders, ShowHint, Showing, SmallImages, StateImages, TabStop, Tag, Top, TopItem, ViewOrigin, ViewStyle, Visible, VisibleRowCount e Width

Principais Métodos

Arrange, Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen, ContainsControl, ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindCaption, FindComponent, FindData, Focused, Free, GetItemAt, GetNearestItem, GetNextItem, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl, Invalidate, IsEditing, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, Scroll, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show, Sort, StringWidth, Update e UpdateItems

Principais Eventos

OnChange, OnChanging, OnClick, OnColumnClick, OnCompare, OnDblClick, OnDeletion, OnDragDrop, OnDragOver, OnEdited, OnEditing, OnEndDrag, OnEnter, OnExit, OnInsert, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp e OnStartDrag

TMainMenu

Descrição

Esse componente representa uma barra de menu em um formulário.

Unit

Na VCL:

Menus

Na CLX:

QMenus

Principais Propriedades

AutoMerge, ComponentIndex, Images, Items, Name, Owner e Tag

Principais Métodos

FindItem, Free, GetHelpContext, HelpKeyword, HelpTypeMerge e UnMerge

Principais Eventos

Esse componente não possui eventos associados.

TMaskEdit

Descrição

O controle TMaskEdit consiste em uma caixa de edição especial, na qual o usuário só pode digitar um conjunto de caracteres definidos como válidos.

Unit

Na VCL:

Mask

Na CLX:

QMask

Principais Propriedades

Align, AutoSelect, AutoSize, BorderStyle, CharCase, Color, ComponentIndex, Ctl3D, Cursor, DragCursor, DragMode, EditMask, EditText, Enabled, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, IsMasked, Left, MaxLength, Modified, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PassWordChar, ReadOnly, SelLengh, SelStart, SelText, ShowHint, Showing, TabOrder, TabStop, Tag, Text, Top, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, Clear, ClearSelection, ClientToScreen, CopyToClipBoard, CutToClipBoard, Dragging, EndDrag, Focused, GetSelTextBuf, GetTextBuf, GetTextLen, Hide, Invalidate, PasteFromClipBoard, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SelectAll, SendToBack, SetBounds, SetFocus, SetSelTextBuf, SetTextBuf, Show, Update e ValidateEdit

Principais Eventos

OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp

TMEDIAPLAYER

Descrição

O componente TMediaPlayer controla dispositivos que dispõem de um driver MCI (Media Control Interface). O componente consiste em um conjunto de botões (Play, Stop, Eject, Next, Prev, Step, Back e Record), que controla dispositivos multimídia, como CD-ROM, VCR, etc.

Unit

Na VCL:

```
MPlayer
```

Principais Propriedades

```
Align, AutoEnecle, AutoOpen, AutoRewind, BoundsRect, Capabilities, ColredButtons, ComponentIndex,
Cursor, DeviceID, DeviceType, Display, DisplayRect, Enabled, EnabledButtons, EndPos, Error,
ErrorMessage, FileName, Frame, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Mode, Name,
Notify, NotifyValue, Owner, Parent, ParentShowHint, Position, Shareable, ShowHint, Showing, Start,
StartPos, TabOrder, TabStop, Tag, TimeFormat, Top, TrackLength, TrackPosition, Tracks, Visible,
VisibleButtons, Wait e Width
```

Principais Métodos

```
Back, BeginDrag, BringToFront, CanFocus, ClientToScreen, Close, Dragging, Eject, EndDrag, Focused,
GetTextBuf, GetTextLen, Hide, Invalidate, Next, Open, Pause, PauseOnly, Play, Previous, Refresh,
Repaint, Resume, Rewind, ScaleBy, Save, ScreenToClient, SendToBack, SetBounds, SetFocus,
SetSelTextBuf, Show, ShowRecording, Step, Stop e Update
```

Principais Eventos

```
OnClick, OnEnter, OnExit, OnNotify e OnPostClick
```

TMEMO

Descrição

O controle TMemo exibe texto para o usuário e permite que o usuário exiba e digite dados no controle. Ao contrário do controle TEdit, esse controle permite texto de múltiplas linhas.

Unit

Na VCL:

```
StdCtrls
```

Na CLX:

```
QStdCtrls
```

Principais Propriedades

```
Align, Alignment, BorderStyle, Color, ComponentIndex, Ctl3D, Cursor, DragCursor, DragMode, Enabled,
Font, Height, HelpContext, HelpKeyword, HelpTypeHideSelection, Hint, Left, Lines, MaxLength,
Modified, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu,
ReadOnly, ScrollBars, SelLengh, SelStart, SelText, ShowHint, Showing, TabOrder, TabStop, Tag, Text,
Top, Visible, WantReturns, WantTabs, Width e WordWrap
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, Clear, ClearSelection, ClientToScreen, CopyToClipboard, Create,
CutToClipboard, Dragging, EndDrag, Focused, GetSelTextBuf, GetTextBuf, GetTextLen, Hide, Invalidate,
PastFromClipboard, Refresh, RemoveComponent, Repaint, ScaleBy, ScreenToClient, ScrollBy, SelectAll,
SendToBack, SetBounds, SetFocus, SetSelTextBuf, SetTextBuf, Show e Update
```

Principais Eventos

```
OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown,
OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp
```

TMemoBuf

Unit

Na VCL:

RpMemo Na CLX:

QRpMemo

Descrição

Fornece a funcionalidade de textos de múltiplas linhas em componentes do Rave Reports.

Principais Propriedades

BaseReport, Buffer, BufferInc, Field, Justify, MaxSize, Memo, NoCRLF, NoNewLine, Pos, PrintEnd, PrintStart, RichEdit, RTFField, RTFText, Size, Text, Version

Principais Métodos

Append, AppendMemoBuf, ConstraintHeightLeft, Delete, Empty, FreeSaved, InsertMemoBuf, Insert, LoadFromFile, LoadFromStream, MemoHeightLeft, MemoLinesLeft, PrintHeight, PrintLines, ReplaceAll, Reset, RestoreBuffer, RestoreState, RTFLoadFromFile, RTFLoadFromStream, SaveBuffer, SaveState, SaveToStream, SearchFirst, SearchNext, SetData

Principais Eventos

Esta classe não tem eventos associados.

TMemoField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena uma informação de texto em um conjunto arbitrário de bytes cujo tamanho não é predefinido.

Unit

Na VCL e na CLX:

DB

Principais Propriedades

Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsSQLTimeStamp, AsString, Calculated, CanModify, DataSet, DataSize, DataType, DisplayLabel, DisplayName, DisplayText, DisplayWisth, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, Name, Owner, ReadOnly, Required, Size, Tag, Text, Transliterate e Visible

Principais Métodos

Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar, LoadFromFile, LoadFromStream, SaveToFile, SaveToStream e SetData

Principais Eventos

OnChange, OnGetText, OnSetText e OnValidate

TMenuItem

Descrição

Esse componente representa um item de menu.

Unit

Na VCL:

Menus

Na CLX:

QMenus

Principais Propriedades

```
Break, Caption, Checked, Command, ComponentIndex, Count, Enabled, GroupIndex, HelpContext,
HelpKeyword, HelpTypeHint, Items, Name, Owner, Parent, ShortCut, Tag e Visible
```

Principais Métodos

```
Add, Click, IndexOf, Insert e Remove
```

Principais Eventos

```
OnClick
```

TMetafile

Descrição

Esse objeto representa um gráfico no formato Metafile do Windows (arquivo no formato *.WMF).

Unit

Na VCL:

```
Graphics
```

Principais Propriedades

```
Handle, Height, Inch, Empty e Width
```

Principais Métodos

```
Assign, ClassName, ClassParent, ClassType, Create, Destroy, Free, LoadFromFile e SaveToFile
```

Principais Eventos

```
OnChange
```

TMIDASConnection

Descrição

Esse componente é responsável por manter uma conexão a um servidor remoto em uma aplicação multicamada.

Unit

Na VCL:

```
MIDAScon
```

Principais Propriedades

```
ConnectType, ServerPort, UseBroker, ComputerName, Connected, AppServer, LoginPrompt, ObjectBroker,
ServerGUID, ServerName
```

Principais Métodos

```
Create, Destroy e GetProvider
```

Principais Eventos

```
OnGetUserName, OnLogin, AfterConnect, AfterDisconnect, BeforeConnect, BeforeDisconnect
```

TMidasPageProducer

Descrição

Esse componente gera uma string de comandos HTML a partir de um template, retornando dados acessados por uma aplicação servidora.

Unit

Na VCL:

```
Midprod
```

Principais Propriedades

ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DesignInfo, Dispatcher, EnableXMLIslands, HTMLDoc, HTMLFile, IncludePathURL, Name, Owner, StripParamQuotes, Styles, StylesFile, Tag, VCLComObject, WebPageItems

Principais Métodos

AfterConstruction, Assign, BeforeDestruction, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, Contentm ContentFromStream, ContentFromString, Create, DefaultHandler, Destroy, DestroyComponents, Destroying, Dispatch, ExecuteAction, FieldAddress, FindComponent, FindXMLBroker, FindXMLBrokerName, Free, FreeInstance, FreeNotification, FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GetXMLData, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName, NewInstance, RemoveComponent, SafeCallException, SetIncludePathURL, SetStyles, SetStylesFile, UpdateAction

Principais Eventos

OnAfterGetContent, OnAfterGetXMLData, OnBeforeGetContent, OnBeforeGetXMLData, OnHTMLTag

TNMDayTime

Descrição

Esse componente obtém a data e a hora de um servidor Internet.

Unit

Na VCL:

NMDayTim

Principais Propriedades

DayTimeStr, Host, Port, ReportLevel, Status, TimeOut

Principais Métodos

Esse componente não possui métodos.

Principais Eventos

OnConnect, OnConnectionFailed, OnDisconnect, OnHostResolved, OnInvalidHost, OnStatus

TNMEcho

Descrição

Esse componente envia um texto para o servidor Internet (recebendo-o de volta, como um eco).

Unit

Na VCL:

NMEcho

Principais Propriedades

ElapsedTime, Host, LocalIP, Port, RemoteIP, ReportLevel, Status, TimeOut, WSAInfo

Principais Métodos

Abort, Echo, Connect, Disconnect

Principais Eventos

OnConnect, OnConnectionFailed, OnConnectionRequired, OnDisconnect, OnHostResolved, OnInvalidHost, OnStatus

TNMFinger

Descrição

Esse componente permite a conexão a um servidor Finger da Internet.

Unit

Na VCL:

NMFinger

Principais Propriedades

FingerStr, Host, LocalIP, Port, RemoteIP, ReportLevel, TimeOut, User, WSAInfo

Principais Métodos

Esse componente não possui métodos.

Principais Eventos

OnConnect, OnConnectionFailed, OnConnectionRequired, OnDisconnect, OnHostResolved, OnInvalidHost, OnStatus

TNMFTP

Descrição

Esse componente permite a conexão a um servidor FTP da Internet.

Unit

Na VCL:

NMFtp

Principais Propriedades

BytesRecvd, BytesSent, BytesTotal, CurrentDir, FTPDirectoryList, Host, LocalIP, ParseList, Password, Port, Proxy, ProxyPort, RemoteIP, ReplyNumber, ReportLevel, Status, TimeOut, TransactionReply, UserID, Vendor, WSAInfo

Principais Métodos

Abort, Allocate, ChangeDir, Connect, Delete, Disconnect, DoCommand, Download, DownloadRestore, List, MakeDirectory, Mode, Nlist, Reinitialize, RemoveDir, Rename, Upload, UploadAppend, UploadRestore, UploadUnique

Principais Eventos

OnAuthenticationFailed, OnConnect, OnConnectionFailed, OnConnectionRequired, OnDisconnect, OnFailure, OnError, OnHostResolved, OnInvalidHost, OnListItem, OnPacketRecvd, OnPacketSent, OnStatus, OnSuccess, OnTransactionStart, OnTransactionStop, OnUnSupportedFunction

TNMHTTP

Descrição

Esse componente permite a conexão a um servidor HTTP da Internet.

Unit

Na VCL:

NMHtttp

Principais Propriedades

Body, BytesRecvd, BytesSent, BytesTotal, CookieIn, Header, HeaderInfo, Host, InputFileMode, LastErrorNo, LocalIP, OutputFileMode, Port, Proxy, ProxyPort, RemoteIP, ReplyNumber, ReportLevel, SendHeader, Status, TimeOut, TransactionReply, WSAInfo

Principais Métodos

Abort, Copy, Delete, Get, Head, Link, Move, Options, Patch, Post, Trace, UnLink, Wrapped

Principais Eventos

OnAboutToSend, OnConnect, OnConnectionFailed, OnDisconnect, OnFailure, OnHostResolved, OnInvalidHost, OnPacketRecvd, OnPacketSent, OnRedirect, OnStatus, OnSuccess

TNMMSG

Descrição

Esse componente permite o envio de texto ASCII pela Internet utilizando uma conexão TCP/IP.

Unit

Na VCL:

```
NMMsg
```

Principais Propriedades

```
FromName, Host, LocalIP, Port, ReportLevel, Status, TimeOut, WSAInfo
```

Principais Métodos

```
Abort, Create, PostIt
```

Principais Eventos

```
OnConnect, OnConnectionFailed, OnDisconnect, OnHostResolved, OnInvalidHost, OnMessageSent, OnStatus
```

TNMMSGServ

Descrição

Esse componente permite a recepção de texto ASCII enviado pela Internet utilizando o componente TNMMSG.

Unit

Na VCL:

```
NMMsg
```

Principais Propriedades

```
LocalIP, Port, ReportLevel, Status, TimeOut, WSAInfo
```

Principais Métodos

```
Esse componente não possui métodos.
```

Principais Eventos

```
OnClientContact, OnMSG, OnStatus
```

TNMNNTP

Descrição

Esse componente permite o envio e a recepção de artigos para grupos de notícias da Internet.

Unit

Na VCL:

```
NMNNTP
```

Principais Propriedades

```
AttachFilePath, Attachments, Body, BytesRecvd, BytesSent, BytesTotal, CacheMode, Connected,
CurrentArticle, GroupList, Header, HeaderRecord, HiMessage, Host, LocalIP, LoMessage, NewsDir,
ParseAttachments, Password, Port, PostBody, PostHeader, PostRecord, Posting, RemoteIP, ReplyNumber,
ReportLevel, SelectedGroup, Status, TimeOut, TransactionReply, UserId, WSAInfo
```

Principais Métodos

```
Abort, Connect, Disconnect, GetArticle, GetArticleBody, GetArticleHeader, GetArticleList,
GetGroupList, PostArticle e SetGroup
```

Principais Eventos

```
OnAbort, OnArticle, OnArticleCacheUpdate, OnAuthenticationFailed, OnAuthenticationNeeded, OnBody,
OnBodyCacheUpdate, OnConnect, OnConnectionFailed, OnDisconnect, OnGroupListCacheUpdate,
OnGroupListUpdate, OnGroupSelect, OnGroupSelectRequired, OnHeader OnHeaderCacheUpdate, OnHeaderList,
OnHostResolved, OnInvalidArticle, OnInvalidHost, OnPacketRecvd, OnPacketSent, OnPosted,
OnPostFailed, OnStatus
```

TNMPOP3

Descrição

Esse componente permite o recebimento de e-mails de um servidor POP3 da Internet.

Unit

Na VCL:

NMPOP3

Principais Propriedades

AttachFilePath, BytesRecvd, BytesTotal, Connected, DeleteOnRead, Host, LocalIP, MailCount, MailMessage, Password, Port, RemoteIP, ReportLevel, Status, Summary, TimeOut, TransactionReply, UserID, WSAInfo

Principais Métodos

Abort, Connect, DeleteMailMessage, Disconnect, GetMailMessage, GetSummary, List, Reset

Principais Eventos

OnAuthenticationFailed, OnAuthenticationNeeded, OnConnect, OnConnectionFailed, OnConnectionRequired, OnHostResolved, OnStatus, OnDisconnect, OnFailure, OnInvalidHost, OnList, OnPacketRecvd, OnReset, OnRetrieveEnd, OnRetrieveStart e OnSuccess

TNMUUPROCESSOR

Descrição

Esse componente permite codificar e decodificar arquivos no formato MIME.

Unit

Na VCL:

NMUUP

Principais Propriedades

InputStream, Methos e OutputStream

Principais Métodos

Encode e Decode

Principais Eventos

OnBeginEncode, OnEndEncode, OnBeginDecode, OnEndDecode e OnError

TNMSMTP

Descrição

Esse componente permite enviar e-mails pela Internet.

Unit

Na VCL:

NMSMTP

Principais Propriedades

ClearParams, EncodeType, FinalHeader, Host, LocalIP, Port, PostMessage, ReplyNumber, Status, TimeOut, TransactionReply, UserID e WSAInfo

Principais Métodos

Abort, ClearParameters, Connect, Disconnect, ExpandList, SendMail, Verify

Principais Eventos

OnAttachmentNotFound, OnAuthenticationFailed, OnConnect, OnConnectionFailed, OnConnectionRequired, OnDisconnect, OnEncodeEnd, OnEncodeStart, OnFailure, OnHeaderIncomplete, OnHostResolved, OnInvalidHost, OnMailListReturn, OnRecipientNotFound, OnSendStart, OnStatus

TNMSTRM

Descrição

Esse componente permite enviar streams pela Internet.

Unit

Na VCL:

```
NMSTRM
```

Principais Propriedades

```
FromName, Host, LocalIP, Port, ReportLevel, Status, TimeOut, WSAInfo
```

Principais Métodos

```
Create e PostIt
```

Principais Eventos

```
OnConnect, OnConnectionFailed, OnDisconnect, OnHostResolved, OnStatus, OnInvalidHost, OnMessageSent,
OnPacketSent
```

TNMSTRMServ

Descrição

Esse componente permite receber streams pela Internet, enviadas pelo componente

```
TNMSTRM.
```

Unit

Na VCL:

```
NMSTRM
```

Principais Propriedades

```
LocalIP, Port, ReportLevel, Status, TimeOut, WSAInfo
```

Principais Métodos

```
Esse componente não possui métodos.
```

Principais Eventos

```
OnMSG, OnClientContact, OnStatus
```

TNMTime

Descrição

Esse componente obtém a data e a hora de um servidor Internet, pelo protocolo RFC 868.

Unit

Na VCL:

```
NMTime
```

Principais Propriedades

```
Host, Port, ReportLevel, Status, TimeOut, TimeStr
```

Principais Métodos

```
Create
```

Principais Eventos

```
OnConnect, OnConnectionFailed, OnDisconnect, OnHostResolved, OnInvalidHost, OnStatus
```

TNMUDP

Descrição

Esse componente permite o envio de pacotes pela Internet usando o protocolo UDP.

Unit

Na VCL:

```
NMUDP
```

Principais Propriedades
LocalHost, RemoteHost, RemoteLevel e RemotePort

Principais Métodos
Create, ReadBuffer, ReadStream, SendBuffer, SendStream

Principais Eventos
OnBufferInvalid, OnDataSend, OnInvalidHost, OnDataReceived, OnStatus, OnStreamInvalid

TNMGeneralServer

Descrição

Esse componente é, na realidade, a classe-base de muitos outros componentes para acesso a Internet do Delphi.

Unit

Na VCL:

PSock

Principais Propriedades
Esse componente não possui propriedades.

Principais Métodos
CaptureFile, CaptureStream, CaptureString, Read, ReadLn, SendBuffer, SendFile, SendStream, Serve, Transaction, Write, Writeln

Principais Eventos
OnClientContact

TNMURL

Descrição

Esse componente converte strings em URLs e vice-versa.

Unit

Na VCL:

NMUrl

Principais Propriedades
Decode, Encode e InputString

Principais Métodos
Esse componente não possui métodos.

Principais Eventos
OnError

TNotebook

Descrição

O componente TNotebook é um componente que pode exibir muitas páginas, cada uma com os seus próprios controles.

Unit

Na VCL:

ExtCtrls

Principais Propriedades
ActivePage, Align, BoundsRect, Color, ComponentCount, ComponentIndex, Components, ControlCount, Controls, Ctl3D, Cursor, DragCursor, DragMode, Enabled, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeLeft, Name, Owner, PageIndex, Pages, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu, ShowHint, Showing, TabOrder, TabStop, Tag, Top, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, Clear, ClientToScreen, ContainsControl, Create, Destroy, Dragging, EndDrag, FindComponent, Focused, Free, GetTextBuf, GetTextLen, Hide, InsertComponent, InsertControl, Invalidate, Refresh, RemoveComponent, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnMouseDown, OnMouseMove e OnMouseUp

TOLEContainer

Descrição

O componente TOLEContainer faz a ligação entre a sua aplicação e um servidor OLE.

Unit

Na VCL:

OleCtrls

Principais Propriedades

Active, Align, AllowInPlace, AutoActivate, AutoSize, BorederStyle, BoundsRect, ComponentIndex, ConvertDlgHelp, Ctl3D, Cursor, DragCursor, DragMode, Enabled, Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, InPlaceActive, Left, Modified, Name, ObjClass, ObjDoc, ObjItem, Owner, Parent, ParentCtl3D, PrintInfo, ShowHint, Showing, Storage, TabOrder, TabStop, Tag, Top, Visible, Width e Zoom

Principais Métodos

BeginDrag, BringToFront, CanFocus, ClientToScreen, CopyToClipboard, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, Invalidate, LoadFromFile, OLEObjAllocated, Refresh, Repaint, SaveToFile, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetTextBuf, Show e Update

Principais Eventos

OnActivate, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp e OnStatusLineEvent

TOpenDialog

Descrição

O controle TOpenDialog fornece uma caixa de diálogo para a abertura de arquivos.

Unit

Na VCL:

Dialogs

Na CLX:

QDialogs

Principais Propriedades

ComponentIndex, Ctl3D, DefaultExt, FileEditStyle, FileName, Files, Filter, FilterIndex, HelpContext, HelpKeyword, HelpTypeHistoryList, InitialDir, Name, Options, Owner, Tag e Title

Principais Métodos

Execute

Principais Eventos

Esse componente não possui eventos associados.

TOpenPictureDialog

Descrição

O controle TOpenPictureDialog fornece uma caixa de diálogo para a abertura de arquivos que armazenam imagens.

Unit

Na VCL:

`Dialogs`

Principais Propriedades

`ComponentIndex, Ctl3D, DefaultExt, FileEditStyle, FileName, Files, Filter, FilterIndex, HelpContext, HelpKeyword, HelpTypeHistoryList, InitialDir, Name, Options, Owner, Tag e Title`

Principais Métodos

`Execute`

Principais Eventos

Esse componente não possui eventos associados.

TOUTLINE

Descrição

O controle TOutline é usado para exibir dados em vários níveis hierárquicos.

Unit

Na VCL:

`Outline`

Principais Propriedades

`Align, BorderStyle, BoundsRect, Canvas, Color, ComponentIndex, Ctrl3D, Cursor, Directory, DragCursor, DragMode, Enabled, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, ItemCount, ItemHeight, Items, ItemSeparator, Left, Lines, Name, Options, OutlineStyle, Owner, Parent, ParentColor, ParentCtrl3D, ParentFont, ParentShowHint, PictureClosed, PictureLeaf, PictureMinus, PictureOpen, PicturePlus, PopupMenu, Row, ScrollBars, SelectedItem, ShowHint, Showing, Style, TabOrder, TabStop, Tag, Top, Visible e Width`

Principais Métodos

`Add, AddChild, AddChildObject, AddObject, BeginDrag, BeginUpdate, BringToFront, CanFocus, Clear, ClientToScreen, Dragging, EndDrag, EndUpdate, Focused, FullCollapse, FullExpand, GetDataItem, GetItem, GetTextBuf, GetTextItem, GetTextLen, Hide, Insert, InsertObject, Invalidate, LoadFromFile, Refresh, Repaint, SaveToFile, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, SetUpdateState, Show e Update`

Principais Eventos

`OnClick, OnCollapse, OnDblClick, OnDragDrop, OnDragOver, OnDrawItem, OnEndDrag, OnEnter, OnExit, OnExpand, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp`

TOUTLINENODE

Descrição

Esse objeto contém um item de um componente TOutline.

Unit

Na VCL:

`Outline`

Principais Propriedades

`Data, Expanded, FullPath, HasItems, Index, IsVisible, Level, Parent, Text e TopItem`

Principais Métodos

`ChangeLevelBy, ClassName, ClassParent, ClassType, Create, Destroy, Expand, Free, FullExpand, GetFirstChild, GetLastChild, GetNextChild, GetPrevChild e MoveTo`

Principais Eventos

Esse objeto não possui eventos associados.

TPageControl

Descrição

O componente TPageControl é um componente que pode exibir muitas páginas, cada uma com os seus próprios controles.

Unit

Na VCL:

```
ComCtrls
```

Na CLX:

```
QComCtrls
```

Principais Propriedades

```
ActivePage, Align, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect, ClientWidth, Color,
ComponentCount, ComponentIndex, Components, ControlCount, Controls, Cursor, DragCursor, DragMode,
Enabled, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, MultiLine, Name, Owner,
PageIndex, Pages, Parent, ParentColor, ParentFont, ParentShowHint, PopupMenu, ShowHint, Showing,
TabHeight, TabOrder, TabStop, TabWidth, Tag, Top, Visible e Width
```

Principais Métodos

```
Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen, ContainsControl,
ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindComponent, FindNextPage, Focused, Free,
GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl, Invalidate,
Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, ScrollBy, SelectNextPage,
SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnChange, OnChanging, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnMouseDown, OnMouseMove,
OnMouseUp e OnStartDrag
```

TPageProducer

Descrição

Esse componente gera uma string de comandos HTML a partir de um template.

Unit

Na VCL e na CLX:

```
HTTPProd
```

Principais Propriedades

```
ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DesignInfo,
Dispatcher, HTMLDoc, HTMLFile, Name, Owner, StripParamQuotes, Tag, VCLComObject
```

Principais Métodos

```
AfterConstruction, Assign, BeforeDestruction, ClassInfo, ClassName, ClassNameIs, ClassParent,
ClassType, CleanupInstance, Content, ContentFromStream, ContentFromString, Create, DefaultHandler,
Destroy, DestroyComponents, Destroying, Dispatch, ExecuteAction, FieldAddress, FindComponent, Free,
FreeInstance, FreeNotification, FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable,
GetNamePath, GetParentComponent, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize,
MethodAddress, MethodName, NewInstance, RemoveComponent, SafeCallException, UpdateAction
```

Principais Eventos

```
OnHTMLTag
```

TPaintBox

Descrição

Esse componente fornece uma área retangular na qual a sua aplicação pode exibir um desenho em um formulário.

Unit

Na VCL:

```
ExtCtrls
```

Na CLX:

```
QExtCtrls
```

Principais Propriedades

```
Align, BoundsRect, Canvas, ComponentIndex, Color, Cursor, DragCursor, DragMode, Enabled, Font,
Height, Hint, Left, Name, Owner, Parent, ParentColor, ParentFont, ParentShowHint, PopupMenu,
ShowHint, Tag, Top, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen,
Invalidate, Refresh, Repaint, ScreenToClient, SendToBack, SetBounds, SetTextBuf e Update
```

Principais Eventos

```
OnClick, OnDblClick, OnDragDrop, OnDragOver, OnMouseDown, OnMouseMove, OnMouseUp e OnPaint
```

TPanel

Descrição

O componente TPanel permite que se coloquem painéis em formulários, nos quais podem ser inseridos outros controles, criando barras de ferramentas e barras de status.

Unit

Na VCL:

```
ExtCtrls
```

Na CLX:

```
QExtCtrls
```

Principais Propriedades

```
Align, Alignment, Bevelinner, BevelOuter, BevelWidth, BorderStyle, BorderWidth, BoundsRect, Caption,
Color, ComponentCount, ComponentIndex, Components, ControlCount, Controls, Ctl3D, Cursor,
DockClientCount, DockClients, DragCursor, DragMode, Enabled, Font, Height, HelpContext, HelpKeyword,
HelpTypeHint, Left, Locked, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont,
ParentShowHint, PopupMenu, ShowHint, Showing, TabOrder, TabStop, Tag, Top, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, ClientToScreen, ContainsControl, Dragging, EndDrag,
FindComponent, Focused, GetTextBuf, GetTextLen, Hide, InsertComponent, InsertControl, Invalidate,
Refresh, RemoveComponent, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds,
SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnMouseDown, OnMouseMove, OnMouseUp e OnResize
```

TPen

Descrição

Esse objeto é usado para desenhar linhas em um objeto do tipo TCanvas.

Unit

Na VCL:

```
Graphics
```

Na CLX:

```
QGraphics
```

Principais Propriedades
Color, Handle, Mode, Style e Width

Principais Métodos
Assign, ClassName, ClassParent, ClassType, Create, Destroy e Free

Principais Eventos
OnChange

TPicture

Descrição

Esse objeto representa um bitmap, um ícone ou um gráfico no formato Metafile do Windows.

Unit

Na VCL:

Graphics

Na CLX:

QGraphics

Principais Propriedades
Bitmap, Graphic, Icon, Height, Metafile e Width

Principais Métodos
Assign, ClassName, ClassParent, ClassType, Create, Destroy, Free, LoadFromFile e SaveToFile

Principais Eventos
OnChange

TPopupMenu

Descrição

Esse componente representa um menu flutuante (pop-up).

Unit

Na VCL:

Menus

Na CLX:

QMenus

Principais Propriedades
Alignment, AutoPopup, ComponentCount, ComponentIndex, Components, Handle, HelpContext, HelpKeyword, HelpTypeImages, Name, Owner, PopupComponent e Tag

Principais Métodos
FindComponent, FindItem, Free e Popup

Principais Eventos
OnPopup

TPrintDialog

Descrição

O controle TPrintDialog fornece uma caixa de diálogo de impressão na qual o usuário pode selecionar a impressora, o número de cópias, as páginas a serem impressas, se deve ser usada a opção de cópias agrupadas, etc.

Unit

Na VCL:

```
Dialogs
```

Principais Propriedades

```
Collate, ComponentIndex, Copies, FromPage, HelpContext, HelpKeyword, HelpTypeMaxPage, MinPage, Name,
Options, Owner, PrintRange, PrintToFile, Tag e ToPage
```

Principais Métodos

```
Execute
```

Principais Eventos

```
Esse componente não possui eventos associados.
```

TPrinter

Descrição

Esse objeto representa a impressora corrente do sistema.

Unit

Na VCL:

```
Printers
```

Na CLX:

```
QPrinters
```

Principais Propriedades

```
Aborted, Canvas, Fonts, Handle, Orientation, PageHeight, PageWidth, PageNumber, PrinterIndex,
Printing, Printers e Title
```

Principais Métodos

```
Abort, BeginDoc, ClaassName, ClassParent, ClassType, Create, Destroy, EndDoc, Free, GetPrinter,
NewPage e SetPrinter
```

Principais Eventos

Esse componente não possui eventos associados.

TPrinterSetupDialog

Descrição

O controle TPrinterSetupDialog fornece uma caixa de diálogo de configuração de impressora na qual o usuário pode configurar a impressora a ser usada.

Unit

Na VCL:

```
Dialogs
```

Principais Propriedades

```
ComponentIndex, HelpContext, HelpKeyword, HelpTypeName, Owner e Tag
```

Principais Métodos

```
Execute
```

Principais Eventos

Esse componente não possui eventos associados.

TProgressBar

Descrição

O controle TProgressBar É um componente que pode ser utilizado para exibir o progresso da execução de uma tarefa.

Unit

Na VCL:

```
ComCtrls
```

Na CLX:

```
QComCtrls
```

Principais Propriedades

```
Align, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect, ClientWidth, ComponentCount,
ComponentIndex, Components, ControlCount, Controls, Cursor, Enabled, Handle, Height, HelpContext,
HelpKeyword, HelpTypeHint, Left, Max, Min, Name, Owner, Parent, ParentShowHint, PopupMenu, Position,
ShowHint, Showing, Step, TabOrder, TabStop, Tag, Top, Visible e Width
```

Principais Métodos

```
Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen,
ContainsControl, ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindComponent, Focused,
Free, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl,
Invalidate, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, SendToBack,
SetBounds, SetFocus, SetTextBuf, Show Stepit, StepBy e Update
```

Principais Eventos

```
OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnMouseDown, OnMouseMove e OnMouseUp e OnStartDrag
```

TProvider

Descrição

Esse componente é responsável por fornecer os dados de um dataset à aplicação-cliente.

Unit

Na VCL:

```
Provider
```

Principais Propriedades

```
DataSet, UpdateMode, Options, ResolveToDataSet, Resolver, Constraints, Data, Provider
```

Principais Métodos

```
ApplyUpdates, Create, FetchData, Reset, SetParams, Destroy, GetRecords, DataRequest, GetMetaData
```

Principais Eventos

```
OnGetDataSetProperties, AfterUpdateRecord, BeforeUpdateRecord, OnGetData, OnUpdateData,
OnUpdateError, OnDataRequest
```

TQRDBText

Descrição

Esse controle é usado para exibir, em um relatório, o valor de um campo de um banco de dados.

Unit

Na VCL:

```
QRCtrls
```

Principais Propriedades

```
Alignment, AutoSize, Color, Cursor, DataField, DataSource, Font, Height, Hint, Left, Name,
ParentFont, Tag, Top, Transparent, Visible e Width
```

Principais Métodos

Esse controle não possui métodos associados.

Principais Eventos

```
OnPrint
```

TQRLABEL

Descrição

O controle TQRLabel é usado na exibição de texto como cabeçalho de coluna em um relatório.

Unit

Na VCL:

```
QRCtrls
```

Principais Propriedades

```
Alignment, AlignToBand, AutoSize, Caption, Color, Cursor, Font, Height, Hint, Left, Name,
ParentFont, Tag, Top, Transparent, Visible e Width
```

Principais Métodos

Esse controle não possui métodos associados.

Principais Eventos

```
OnPrint
```

TQRMEMO

Descrição

Esse controle é usado para inserir um texto com múltiplas linhas em um relatório.

Unit

Na VCL:

```
QRCtrls
```

Principais Propriedades

```
Alignment, Color, Cursor, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Lines, Name,
ParentFont, TabStop, Tag, Top, Width e WordWrap
```

Principais Métodos

Esse controle não possui métodos associados.

Principais Eventos

Esse controle não possui eventos associados.

TQRPREVIEW

Descrição

Esse controle facilita a criação de formulários para a pré-visualização de páginas de relatórios.

Unit

Na VCL:

```
QRCtrls
```

Principais Propriedades

```
Align. AutoScroll, BorderStyle, Color, Ctl3D, Cursor, DragCursor, DragMode, Enabled, Font, Height,
HelpContext, HelpKeyword, HelpTypeHint, HorzScrollBar, Left, Name, PageNumber, ParentColor,
ParentCtl3d, ParentFont, ParentShowHint, PopupMenu, ShowHint, TabOrder, TabStop, Tag, Top,
VertScrollBar, Visible, Width e Zoom
```

Principais Métodos

Show, ZoomToWidth e ZoomToFit

Principais Eventos

OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnMouseDown, OnMouseMove, OnMouseUp e OnResize

TQRPrinter

Descrição

O controle TQRPrinter é usado como um objeto de impressão (representa uma impressora), com capacidades de pré-visualização, leitura e gravação. Um objeto desse tipo, chamado QRPrinter, é sempre criado quando a sua aplicação começa a ser executada. Os objetos do tipo TQuickReport usam esse objeto nas suas tarefas de impressão.

Unit

Na VCL:

QRCtrls

Principais Propriedades

Canceled, Canvas, EnableOpenBtn, EnablePrintBtn, EnableSaveBtn, FromPage, Orientation, Page, PageCount, PageHeight, PageNumber, PageWidth, PrinterOk, ShowProgress, Status, Thumbs, Title e ToPage

Principais Métodos

BeginDoc, Cancel, CleanUp, EndDoc, Load, NewPage, Preview, Print e Save

Principais Eventos

OnPreview

TQRShape

Descrição

Esse controle é usado para exibir figuras em um relatório.

Unit

Na VCL:

QRCtrls

Principais Propriedades

Brush, Cursor, Height, Hint, Left, Name, Pen, Shape, Tag, Top, Visible e Width

Principais Métodos

Esse controle não possui métodos associados.

Principais Eventos

Esse controle não possui eventos associados.

TQRSysData

Descrição

Esse controle é usado para exibir dados do sistema em um relatório.

Unit

Na VCL:

QRCtrls

Principais Propriedades

Alignment, AlignToBand, AutoSize, Color, Cursor, Data, Font, Height, Hint, Left, Name, ParentFont, Tag, Text, Top, Transparent, Visible e Width

Principais Métodos

BeginDoc, Cancel, CleanUp, EndDoc, Load, NewPage, Preview, Print e Save

Principais Eventos

OnPrint

TQUERY

Descrição

Esse componente permite que o Delphi execute declarações SQL no Borland Database Engine ou em um servidor SQL.

Unit

DBTables

Principais Propriedades

Active, AutoCalcFilelds, BOB, CanModify, Database, DatabaseName, DatabaseSource, DBHandle, DBLocale, EOF, FieldCount, FieldDefs, Field, Handle, Local, Locale, Modified, Name, Owner, ParamCount, Params, Prepared, RecordCount, RequestLive, SQL, SQLBinary, State, StmtHandle, Tag, Text, Unidirecional e UpdateMode

Principais Métodos

Append, AppendRecord, Cancel, CheckBrowseMode, ClearFields, Close, CursorPosChanged, Delete, DisableControls, Edit, EnableControls, ExecSQL, FieldByName, FindField, First, FreeBookmark, GetBookmark, GetFieldNames, GotoBookmark, Insert, InsertRecord, Last, MoveBy, Next, Open, ParamByName, Post, Prepare, Prior, Refresh, SetFields, UnPrepared, UpdateCursorPos e UpdateRecord

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, OnCalcFields e OnNewRecord

TQUERYTABLEPRODUCER

Descrição

Esse componente gera uma string de comandos HTML que forma uma tabela com os dados provenientes de uma consulta SQL.

Unit

Na VCL:

DBBDEWeb

Principais Propriedades

DataSet, Query, Caption, CaptionAlignment, Columns, Editor, Footer, Header, MaxRows, RowAttributes e TableAttributes

Principais Métodos

Content, BeginUpdate, Create, Destroy, EndUpdate

Principais Eventos

OnCreateContent, OnFormatCell, OnGetTableCaption

TQUEUE

Descrição

Esta classe permite manipular objetos usando uma estrutura de dados do tipo Fila. Neste tipo de estrutura de dados, o primeiro elemento a entrar é o primeiro a sair, sendo comumente denominada estrutura do tipo FIFO – "First In, First Out".

Unit

```
Contnrs
```

Principais Propriedades

Esta classe não possui propriedades associadas.

Principais Métodos

```
AfterConstruction,. AtLeast, BeforeDestruction, ClassInfo, ClassName, ClassNameIs, ClassParent,
ClassType, CleanupInstance, Count, Create, DefaultHandler, Destroy, Dispatch, FieldAddress, Free,
FreeInstance, GetInterface, GetInterfaceEntry, GetInterfaceTable, InheritsFrom, InitInstance,
InstanceSize, MethodAddress, MethodName, NewInstance, Peek, Pop, Push, SafeCallException
```

Principais Eventos

Esta classe não possui propriedades associadas.

TQuickRep

Descrição

O controle TQuickReport é responsável pela funcionalidade da impressão de dados, transformando formulários em relatórios.

Unit

Na VCL:

```
QuickRpt
```

Principais Propriedades

```
ColumnMarginInches, ColumnMarginsMM, Columns, DataSet, DisplayPrintDialog, LeftMarginInches,
LeftMarginsMM, Name, Orientation, PageCount, PageHeight, PageNumber, PageWidth, RecordCount,
RecordNo, ReportTitle, ReportType, RestartData, ShowProgress, SQLCompatible Tag e TitleBeforeHeader
```

Principais Métodos

```
NewPage, Prepare, Preview e Print
```

Principais Eventos

```
AfterDetail, AfterPrint, BeforeDetail, BeforePrint, OnEndPage, OnFilter, OnNeedData e OnStartPage
```

TRadioButton

Descrição

O controle TRadioButton é um botão de opção exclusiva que permite ao usuário selecionar ou não uma opção no aplicativo.

Unit

Na VCL:

```
StdCtrls
```

Na CLX:

```
QStdCtrls
```

Principais Propriedades

```
Align, Alignment, Caption, Checked, Color, ComponentIndex, Ctl3D, Cursor, DragCursor, DragMode,
Enabled, Edit, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Owner,
Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu, ShowHint, Showing,
TabOrder, TabStop, Tag, Top, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf,
GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack,
SetBounds, SetTextBuf, Show e Update
```

Principais Eventos

OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove e OnMouseUp

TRADIOGROUP

Descrição

O controle TRadioGroup permite a exibição de um conjunto de botões de rádio que representam opções mutuamente exclusivas.

Unit

Na VCL:

ExtCtrls

Na CLX:

QExtCtrls

Principais Propriedades

Align, Caption, Color, Columns, ComponentIndex, Ctl3D, Cursor, DragCursor, DragMode, Enabled, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, ItemIndex, Items, Left, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu, ShowHint, Showing, TabOrder, TabStop, Tag, Top, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, ClientToScreen, ContainsControl, Dragging, EndDrag, FindComponent, Focused, GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus e SetTextBuf

Principais Eventos

OnClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter e OnExit

TRDSCONNECTION

Descrição

Este componente é responsável pela Conexão e acesso Remoto a bancos de dados através do Mecanismo Activex Data Objects (ADO).

Unit

Na VCL:

AdODB

Principais Propriedades

AppServer, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, ComputerName, Connected, DataSetCount, DataSets, DataSpaceObject, DesignInfo, InternetTimeout, LoginPrompt, Name, Owner, ServerName, StreamedConnected, Tag, VCLComObject

Principais Métodos

AfterConstruction, Assign, BeforeDestruction, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, Close, Create, DefaultHandler, Destroy, DestroyComponents, Destroying, Dispatch, DoConnect, DoDisconnect, ExecuteAction, FieldAddress, FindComponent, Free, FreeInstance, FreeNotification, FreeOnRelease, GetConnected, GetDataSet, GetDataSetCount, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GetRecordset, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, Loaded, MethodAddress, MethodName, NewInstance, Open, RegisterClient, RemoveComponent, SafeCallException, SendConnectEvent, SetConnected, UnRegisterClient, UpdateAction

Principais Eventos

AfterConnect, AfterDisconnect, BeforeConnect, BeforeDisconnect, OnLogin.

TRegistry

Descrição

Essa classe representa o registro do Windows, permitindo que uma aplicação possa acessá-lo e alterá-lo.

Unit

Na VCL:

```
Registry
```

Principais Propriedades

```
CurrentKey, CurrentPath, LazyWrite, RootKey
```

Principais Métodos

```
CloseKey, Create, CreateKey, DeleteKey, DeleteValue, Destroy, GetDataInfo, GetDataSize, GetDataType,
GetKeyInfo, GetKeyNames, GetValueNames, HasSubKeys, KeyExists, LoadKey, MoveKey, OpenKey,
OpenKeyReadOnly, ReadBool, ReadCurrency, ReadDate, ReadDateTime, ReadFloat, ReadInteger, ReadString,
ReadTime, RegistryConnect, RenameValue, ReplaceKey, RestoreKey, SaveKey, UnLoadKey, ValueExists,
WriteBinaryData, WriteBool, WriteCurrency, WriteDate, WriteDateTime, WriteExpandString, WriteFloat,
WriteInteger, WriteString, WriteTime
```

Principais Eventos

Essa classe não possui eventos associados.

TRegistryIniFile

Descrição

Essa classe representa o registro do Windows, permitindo que uma aplicação possa acessá-lo e alterá-lo.

Unit

Na VCL:

```
Registry
```

Principais Propriedades

```
RegIniFile, FileName
```

Principais Métodos

```
Creat, DeleteKey, EraseSection, ReadDate, ReadDateTime, ReadFloat, ReadInteger, ReadSection,
ReadSections, ReadSectionValues, ReadString, ReadTime, UpdateFile, WriteDate, WriteDateTime,
WriteFloat, WriteInteger, WriteString, WriteTime, ReadBool, SectionExists, ValueExists, WriteBool
```

Principais Eventos

Essa classe não possui eventos associados.

TRemoteServer

Descrição

Esse componente É responsável por manter uma conexão a um servidor remoto em uma aplicação multicamada.

Unit

Na VCL:

```
MIDAScon
```

Principais Propriedades

```
ComputerName, Connected, AppServer, LoginPrompt, ObjectBroker, ServerGUID, ServerName
```

Principais Métodos

```
Create, Destroy e GetProvider
```

Principais Eventos

OnGetUserName, OnLogin, AfterConnect, AfterDisconnect, BeforeConnect, BeforeDisconnect

TReplaceDialog

Descrição

O controle TReplaceDialog fornece uma caixa de diálogo para pesquisa e substituição de texto.

Unit

Na VCL:

Dialogs

Na CLX:

QDialogs

Principais Propriedades

ComponentIndex, Ctl3D, FindText, HelpContext, HelpKeyword, HelpTypeName, Options, Owner, Replacetext e Tag

Principais Métodos

CloseDialog e Execute

Principais Eventos

OnFind e OnReplace

TRichEdit

Descrição

O controle TRichEdit é semelhante ao controle TMemo, mas permite a aplicação de fontes com diferentes atributos a partes distintas do texto inserido no controle.

Unit

Na VCL:

ComCtrls

Principais Propriedades

Align, Alignment, BorderStyle, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect, ClientWidth, Color, ComponentCount, ComponentIndex, Components, ControlCount, Controls, Ctl3D, Cursor, DragMode, Enabled, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeHideScrollBars, HideSelection, Hint, Left, Lines, MaxLength, Name, Owner, Paragraph, Parent, ParentColor, ParentCtl3D, ParentFont, PlainText, PopupMenu, ReadOnly, ScrollBars, SetAttributes, ShowHint, Showing, TabOrder, TabStop, Tag, Top, Visible, WantTabs, WantReturns, Width e WordWrap

Principais Métodos

Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen, ContainsControl, ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindComponent, FindText, Focused, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl, Invalidate, Print, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp, OnProtectChange, OnResizeRequest, OnSaveClipboard, OnSelectionChange e OnStartDrag

TRpBarsBase

Unit

Na VCL:

RpBars

Na CLX:

```
QRpBars
```

Descrição

Esta classe é a classe-base para a geração de códigos de barra em componentes do Rave Reports.

Principais Propriedades

```
BarBottom, BarCodeJustify, BarCodeRotation, BarHeight, BarTop, BarWidth, BaseReport, Bottom, Center,
CheckSum, CodePage, Extended, ExtendedText, Height, Left, Position, PrintChecksum, PrintReadable,
PrintTop, ReadableHeight, Right, Text, TextJustify, Top, UseChecksum, WideFactor, Width
```

Principais Métodos

```
Create, IsValidChar, Print, PrintFimA, PrintFimB, PrintFimC, PrintXY
```

Principais Eventos

Esta classe não possui eventos associados.

TRpBaseComponent

```
Unit
```

Na VCL:

```
RpBase
```

Na CLX:

```
QRpBase
```

Descrição

Esta classe é a classe-base para todos os components do Rave Reports relacionados com a formatação de saída de relatórios.

Principais Propriedades

```
Version
```

Principais Métodos

Essa classe não possui métodos associados.

Principais Eventos

Essa classe não possui eventos associados.

TRpComponent

```
Unit
```

Na VCL:

```
RpDefine
```

Na CLX:

```
QRpDefine
```

Descrição

Esta classe é a classe-base para todos os componentes do Rave Reports não vinculados à formatação de saída.

Principais Propriedades

```
Version
```

Principais Métodos

Essa classe não possui métodos associados.

Principais Eventos

Essa classe não possui eventos associados.

TRpRender

Unit

Na VCL:

RpRender

Na CLX:

RpRender

Esta é a classe-base para renderização de relatórios.

Principais Propriedades

Active, CacheDir, DisplayName, ImageQuality, MetafileDPI, OnCompress, ServerMode, UseCompression, Version

Essa classe não possui métodos associados.

Principais Eventos

Essa classe não possui eventos associados.

TRpRenderCanvas

Unit

Na VCL:

RpRender

Na CLX:

QRpRender

Descrição

Esta é a classe-base para renderização em tela de relatórios.

Principais Propriedades

Active, CacheDir, DisplayName, ImageQuality, MetafileDPI, OnCompress, ServerMode, UseCompression, Version

Essa classe não possui métodos associados.

Principais Eventos

Essa classe não possui eventos associados.

TRpRenderStream

Unit

Na VCL:

RpRender Na CLX:

RpRenderDescrição

Esta é a classe-base para renderização de relatórios com stream.

Principais Propriedades

Active, CacheDir, DisplayName, ImageQuality, MetafileDPI, OnCompress, ServerMode, UseCompression, Version

Essa classe não possui métodos associados.

Principais Eventos

Essa classe não possui eventos associados.

TRvCustomConnection

```
Unit
```

Na VCL:

```
RpCon
```

Na CLX:

```
QRpCon
```

Descrição

Permite customizar o acesso a dados não provenientes de componentes de acesso a bancos de dados.

Principais Propriedades

```
FieldAliasList, LocalFilter, RuntimeVisibility, Version
```

Principais Métodos

```
WriteBCDData, WriteBlobData, WriteBoolData, WriteCurrData, WriteDateTime, WriteFloatData,
WriteIntData, WriteNullData, WriteStrData
```

Principais Eventos

```
OnFirst, OnGetCols, OnGetRow, OnGetSorts, OnNext, OnOpen, OnRestore, OnSetFilter, OnSetSort, OnValidateRow
```

TRvDatasetConnection

```
Unit
```

Na VCL:

```
RpConDS
```

Na CLX:

```
QRpConDS
```

Descrição

Classe-base para conexão a dados provenientes de componentes de acesso a bancos de dados.

Principais Propriedades

```
DataSet, FieldAliasList, LocalFilter, RuntimeVisibility, Version
```

Principais Métodos

```
WriteBCDData, WriteBlobData, WriteBoolData, WriteCurrData, WriteDateTime, WriteFloatData,
WriteIntData, WriteNullData, WriteStrData
```

Principais Eventos

```
OnFirst, OnGetCols, OnGetRow, OnGetSorts, OnNext, OnOpen, OnRestore, OnSetFilter, OnSetSort,
OnValidateRow
```

TRvNDRWriter

```
Unit
```

Na VCL:

```
RpFiler
```

Na CLX:

```
QRpFiler
```

Descrição

Este componente, em conjuto com TRvRenderPrinter e TRvRenderPreview, é responsável por armazenar um relatório em formato binário até que esteja pronto para impressão ou pré-visualização.

Principais Propriedades

Aborted, AccuracyMethod, AscentHeight, Bins, BKColor, Bold, BottomWaste, BoxLineColor, Canvas, Collate, ColumnEnd, ColumnLinesLeft, ColumnNum, Columns, ColumnStart, ColumnWidth, Copies, CurrentPage, CurrentPass, CursorXPos, CursorYPos, DescentHeight, DeviceName, DevMode, DriverName, Duplex, FileName, FirstPage, FontAlign, FontBaseline, FontBottom, FontCharset, FontColor, FontHandle, FontHeight, FontName, FontPitch, FontRotation, Fonts, FontSize, FontTop, FontWidth, FrameMode, GridVert, Italic, LastPage, LeftWaste, LineBottom, LineHeight, LineHeightMethod, LineMiddle, LineNum, LinesPerInch, LineTop, MacroData, MarginBottom, MarginLeft, MarginRight, MarginTop, MaxCopies, NoBufferLine, NoNTColorFix, NoPrinterPageHeight, NoPrinterPageWidth, Orientation, OriginX, OriginY, OutputInvalid, OutputName, PageHeight, PageInvalid, PageWidth, Papers, PIVar, Port, PrinterIndex, Printers, Printing, ReportDateTime, RightWaste, ScaleX, ScaleY, SectionBottom, SectionLeft, SectionRight, SectionTop, Selection, ShadowDepth, StatusFormat, StatusLabel, StatusText, Stream, StreamMode, Strikeout, Subscript, Superscript, TabColor, TabJustify, TabShade, TextBKMode, Title, TopWaste, TotalPasses, TransparentBitmaps, TruncateText, Underline, Units, UnitsFactor, Version, XDPI, XPos, YDPI, YPos

Principais Métodos

Abort, AbortPage, AdjustLine, AllowAll, AllowPreviewOnly, AllowPrinterOnly, Arc, AssignFont, BrushCopy, CalcGraphicHeight, CalcGraphicWidth, Chord, ClearAllTabs, ClearColumns, ClearTabs, CopyRect, CR, Create, CreateBrush, CreateFont, CreatePen, CreatePoint, CreateRect, Destroy, DrawFocusRect, Draw, Ellipse, Execute, FillRect, Finish, FinishTabBox, FloodFill, FrameRect, GetMemoLine, GetNextLine, GetTab, GotoFooter, GotoHeader, GotoXY, GraphicFieldToBitmap, Home, LF, LinesLeft, LineTo, Macro, MemoLines, MoveTo, NewColumn, NewLine, NewPage, NoPrinters, Pie, Polygon, Polyline, PopFont, PopPos, PopTabs, Print, PrintBitmap, PrintBitmapRect, PrintBlock, PrintCenter, PrintCharJustify, PrintData, PrintDataStream, PrintFooter, PrintHeader, PrintImageRect, PrintJustify, PrintLeft, PrintLn, PrintMemo, PrintRight, PrintTab, PrintXY, PushFont, PushPos, PushTabs, RecoverPrinter, Rectangle, RegisterGraphic, ReleasePrinter, Reset, ResetLineHeight, ResetPrinter, ResetSection, ResetTabs, RestoreFont, RestorePos, RestoreTabs, ReuseGraphic, RoundRect, SaveFont, SavePos, SaveTabs, SelectBin, SelectPaperO SelectPrinter, SetBrush, SetColumns, SetColumnWidth, SetFont, SetPaperSize, SetPen, SetPIVar, SetTab, SetTopOfPage, ShadeToColor, ShowPrintDialog, ShowPrinterSetupDialog, Start, StretchDraw, SupportBin, SupportCollate, SupportDuplex, SupportOrientation, SupportPaper, Tab, TabEnd, TabStart, TabWidth, TextRect, TextWidth, UnregisterGraphic, UpdateStatus, XD2U, XI2D, XI2U, XU2D, XU2I, YD2I, YD2U, YI2D, YI2U, YU2D, YU2I

Principais Eventos

OnAfterPrint, OnBeforePrint, OnDecodeImage, OnNewColumn, OnNewPage, OnPrint, OnPrintFooter, OnPrintHeader, OnPrintPage

TRvProject

Unit

Na VCL:

RpRave

Na CLX:

QRpRave

Descrição

Este componente fornece acesso ao editor visual de relatórios do Rave. Em geral usa-se apenas um componente desta classe em uma aplicação, podendo no entanto usar-se mais de um, se necessário. A sua propriedade ProjectFile define o projeto de relatório (não confundir com o projeto de aplicação Delphi) usado pela aplicação e acessado por este componente. Este arquivo de projeto terá a extensão .RAV e, mesmo constituindo-se em um único arquivo, pode conter a definição de diversos relatórios acessados pela aplicação.

Principais Propriedades

Active, DLLFile, Engine, LoadDesigner, ProjectFile, RaveBlobDateTime, ReportDesc, ReportFullName, ReportName, StoreRAV, Version

Principais Métodos

ClearRaveBlob, Close, Design, DesignReport, Execute, ExecuteReport, GetReportCategoryList, GetReportList, LoadFromFile, LoadFromStream, LoadRaveBlob, Open, ReportDescToMemo, Save, SaveRaveBlob, SaveToFile, SaveToStream, SelectReport, SetParam

Principais Eventos

OnAfterClose, OnAfterOpen, OnBeforeClose, OnBeforeOpen, OnCreate, OnDesignerSave, OnDesignerSaveAs, OnDesignerShow e OnDestroy

TRvRenderPDF

Unit

Na VCL:

RvRenderPDF

Na CLX:

RvRenderPDF

Descrição

Este componente é responsável pela geração de um relatório no formato PDF.

Principais Propriedades

Active, CacheDir, DisplayName, FileExtension, ImageQuality, MetafileDPI, OnCompress, ServerMode, UseCompression, Version

Principais Métodos

Esta classe não possui métodos associados.

Principais Eventos

OnCompress, OnDecodeImage

TRvRenderHTML

Unit

Na VCL:

RvRenderHTML

Na CLX:

RvRenderHTML

Descrição

Este componente é responsável pela geração de um relatório no formato HTML.

Principais Propriedades

Active, CacheDir, DisplayName, FileExtension, ImageQuality, MetafileDPI, OnCompress, ServerMode, UseCompression, Version

Principais Métodos

Esta classe não possui métodos associados.

Principais Eventos

Esta classe não possui eventos associados.

TRvRenderPreview

Unit

Na VCL:

RpRenderPreview

Na CLX:

QRpRenderPreview

Descrição

Esta classe é responsável pela pré-visualização de um relatório a partir de um arquivo gerado pelo componenteTRvNDRWriter.

Principais Propriedades

Aborted, AccuracyMethod, AscentHeight, Bins, BKColor, Bold, BottomWaste, BoxLineColor, Canvas, Collate, ColumnEnd, ColumnLinesLeft, ColumnNum, Columns, ColumnStart, ColumnWidth, Copies, CurrentPage, CurrentPass, CursorXPos, CursorYPos, DescentHeight, DeviceName, DevMode, DriverName, Duplex, FileName, FirstPage, FontAlign, FontBaseline, FontBottom, FontCharset, FontColor, FontHandle, FontHeight, FontName, FontPitch, FontRotation, Fonts, FontSize, FontTop, FontWidth, FrameMode, GridVert, IgnoreFileSettings, Italic, LastPage, LeftWaste, LineBottom, LineHeight, LineHeightMethod, LineMiddle, LineNum, LinesPerInch, LineTop, MacroData, MarginBottom, MarginLeft, MarginMethod, MarginPercent, MarginRight, MarginTop, MaxCopies, Monochrome, NoBufferLine, NoNTColorFix, NoPrinterPageHeight, NoPrinterPageWidth, Orientation, OriginX, OriginY, OutputInvalid, OutputName, PageHeight, PageInc, PageInvalid, Pages, PageWidth, Papers, PIVar, Port, PrinterIndex, Printers, Printing, ReportDateTime, RightWaste, ScaleX, ScaleY, ScrollBox, SectionBottom, SectionLeft, SectionRight, SectionTop, Selection, ShadowDepth, StatusFormat, StatusLabel, StatusText, Stream, StreamMode, Strikeout, Subscript, Superscript, TabColor, TabJustify, TabShade, TextBKMode, Title, TopWaste, TotalPasses, TransparentBitmaps, TruncateText, Underline, Units, UnitsFactor, Version, XDPI, XPos, YDPI, YPos, ZoomFactor, ZoomInc, ZoomPageFactor, ZoomPageWidthFactor

Principais Métodos

Abort, AbortPage, AdjustLine, AllowAll, AllowPreviewOnly, AllowPrinterOnly, Arc, AssignFont, BrushCopy, CalcGraphicHeight, CalcGraphicWidth, Chord, Clear, ClearAllTabs, ClearColumns, ClearTabs, CopyRect, CR, Create, CreateBrush, CreateFont, CreatePen, CreatePoint, CreateRect, Destroy, DrawFocusRect, Draw, Ellipse, Execute, ExecuteCustom, FillRect, Finish, FinishTabBox, FloodFill, FrameRect, GetMemoLine, GetNextLine, GetTab, GotoFooter, GotoHeader, GotoXY, GraphicFieldToBitmap, Home, LF, LinesLeft, LineTo, Macro, MemoLines, MoveTo, NewColumn, NewLine, NewPage, NextPage, NoPrinters, Pie, Polygon, Polyline, PopFont, PopPos, PopTabs, PrevPage, Print, PrintBitmap, PrintBitmapRect, PrintBlock, PrintCenter, PrintCharJustify, PrintData, PrintDataStream, PrintFooter, PrintHeader, PrintImageRect, PrintJustify, PrintLeft, PrintLn, PrintMemo, PrintPage, PrintRight, PrintTab, PrintXY, PushFont, PushPos, PushTabs, RecoverPrinter, Rectangle, RedrawPage, RegisterGraphic, ReleasePrinter, Reset, ResetLineHeight, ResetPrinter, ResetSection, ResetTabs, RestoreFont, RestorePos, RestoreTabs, ReuseGraphic, RoundRect, SaveFont, SavePos, SaveTabs, SelectBin, SelectPaperO SelectPrinter, SetBrush, SetColumns, SetColumnWidth, SetFont, SetPaperSize, SetPen, SetPIVar, SetTab, SetTopOfPage, ShadeToColor, ShowPrintDialog, ShowPrinterSetupDialog, Start, StretchDraw, SupportBin, SupportCollate, SupportDuplex, SupportOrientation, SupportPaper, Tab, TabEnd, TabStart, TabWidth, TextRect, TextWidth, UnregisterGraphic, UpdateStatus, XD2I, XD2U, XI2D, XI2U, XU2D, XU2I, YD2I, YD2U, YI2D, YI2U, YU2D, YU2I, ZoomIn, ZoomOut

Principais Eventos

OnAfterPrint, OnBeforePrint, OnDecodeImage, OnNewColumn, OnNewPage, OnPageChange, OnPrint, OnPrintFooter, OnPrintHeader, OnPrintPage, OnZoomChange

TRvRenderPrinter

Unit

Na VCL:

RpRenderPrinter

Na CLX:

QRpRenderPrinter

Descrição

Esta classe é responsável pela impressão de um relatório a partir de um arquivo gerado pelo componenteTRvNDRWriter

Principais Propriedades

Aborted, AccuracyMethod, AscentHeight, Bins, BKColor, Bold, BottomWaste, BoxLineColor, Canvas, Collate, ColumnEnd, ColumnLinesLeft, ColumnNum, Columns, ColumnStart, ColumnWidth, Copies, CurrentPage, CurrentPass, CursorXPos, CursorYPos, DescentHeight, DeviceName, DevMode, DriverName, Duplex, FileName, FirstPage, FontAlign, FontBaseline, FontBottom, FontCharset, FontColor, FontHandle, FontHeight, FontName, FontPitch, FontRotation, Fonts, FontSize, FontTop, FontWidth, FrameMode, GridVert, IgnoreFileSettings, Italic, LastPage, LeftWaste, LineBottom, LineHeight, LineHeightMethod, LineMiddle, LineNum, LinesPerInch, LineTop, MacroData, MarginBottom, MarginLeft, MarginRight, MarginTop, MaxCopies, NoBufferLine, NoNTColorFix, NoPrinterPageHeight, NoPrinterPageWidth, Orientation, OriginX, OriginY, OutputInvalid, OutputName, PageHeight, PageInvalid, PageWidth, Papers, PIVar, Port, PrinterIndex, Printers, Printing, ReportDateTime, RightWaste, ScaleX, ScaleY, SectionBottom, SectionLeft, SectionRight, SectionTop, Selection, ShadowDepth, StatusFormat, StatusLabel, StatusText, Stream, StreamMode, Strikeout, Subscript, Superscript, TabColor, TabJustify, TabShade, TextBKMode, Title, TopWaste, TotalPasses, TransparentBitmaps, TruncateText, Underline, Units, UnitsFactor, Version, XDPI, XPos, YDPI, YPos

Métodos

Abort, AbortPage, AdjustLine, AllowAll, AllowPreviewOnly, AllowPrinterOnly, Arc, AssignFont, BrushCopy, CalcGraphicHeight, CalcGraphicWidth, Chord, ClearAllTabs, ClearColumns, ClearTabs, CopyRect, CR, Create, CreateBrush, CreateFont, CreatePen, CreatePoint, CreateRect, Destroy, DrawFocusRect, Draw, Ellipse, Execute, FillRect, Finish, FinishTabBox, FloodFill, FrameRect, GetMemoLine, GetNextLine, GetTab, GotoFooter, GotoHeader, GotoXY, GraphicFieldToBitmap, Home, LF, LinesLeft, LineTo, Macro, MemoLines, MoveTo, NewColumn, NewLine, NewPage, NoPrinters, Pie, Polygon, Polyline, PopFont, PopPos, PopTabs, Print, PrintBitmap, PrintBitmapRect, PrintBlock, PrintCenter, PrintCharJustify, PrintData, PrintDataStream, PrintFooter, PrintHeader, PrintImageRect, PrintJustify, PrintLeft, PrintLn, PrintMemo, PrintRight, PrintTab, PrintXY, PushFont, PushPos, PushTabs, RecoverPrinter, Rectangle, RegisterGraphic, ReleasePrinter, Reset, ResetLineHeight, ResetPrinter, ResetSection, ResetTabs, RestoreFont, RestorePos, RestoreTabs, ReuseGraphic, RoundRect, SaveFont, SavePos, SaveTabs, SelectBin, SelectPaperO SelectPrinter, SetBrush, SetColumns, SetColumnWidth, SetFont, SetPaperSize, SetPen, SetPIVar, SetTab, SetTopOfPage, ShadeToColor, ShowPrintDialog, ShowPrinterSetupDialog, Start, StretchDraw, SupportBin, SupportCollate, SupportDuplex, SupportOrientation, SupportPaper, Tab, TabEnd, TabStart, TabWidth, TextRect, TextWidth, UnregisterGraphic, UpdateStatus, XD2U, XI2D, XI2U, XU2D, XU2I, YD2I, YD2U, YI2D, YI2U, YU2D, YU2I

Eventos

OnAfterPrint, OnBeforePrint, OnDecodeImage, OnNewColumn, OnNewPage, OnPrint, OnPrintFooter, OnPrintHeader, OnPrintPage

TRvRenderRTF

Unit

Na VCL:

RvRenderRTF

Na CLX:

RvRenderRTF

Descrição

Este componente é responsável pela geração de um relatório no formato RTF.

Principais Propriedades

Active, CacheDir, DisplayName, FileExtension, ImageQuality, MetafileDPI, OnCompress, ServerMode, UseCompression, Version

Principais Métodos

Esta classe não possui métodos associados.

Principais Eventos

Esta classe não possui eventos associados.

TRvRenderText

```
Unit
```

Na VCL:

```
RvRenderText
```

Na CLX:

RvRenderTextDescrição

Este componente é responsável pela geração de um relatório no formato de Texto ASCII.

Principais Propriedades

```
Active, CacheDir, DisplayName, FileExtension, ImageQuality, MetafileDPI, OnCompress, ServerMode,
UseCompression, Version
```

Principais Métodos

Esta classe não possui métodos associados.

Principais Eventos

Esta classe não possui eventos associados.

TRvQueryConnection

Na VCL:

```
RpConBDE
```

Na CLX:

```
QRpConBDE
```

Descrição

Usado para geração de relatórios vinculados a dados provenientes de um componente TQuery.

Principais Propriedades

```
FieldAliasList, LocalFilter, RuntimeVisibility, Query, Version
```

Principais Métodos

```
WriteBCDData, WriteBlobData, WriteBoolData, WriteCurrData, WriteDateTime, WriteFloatData,
WriteIntData, WriteNullData, WriteStrData
```

Principais Eventos

```
OnFirst, OnGetCols, OnGetRow, OnGetSorts, OnNext, OnOpen, OnRestore, OnSetFilter, OnSetSort,
OnValidateRow
```

TRvSystem

```
Unit
```

Na VCL:

```
RpSystem
```

Na CLX:

```
QRpSystem
```

Descrição

Este componente trabalha em conjunto com componentes das classes TRvRenderPreview, TRvRenderPrinter e TRvNDRWriter para enviar um relatório para pré-visualização na tela ou para impressão.

Principais Propriedades

BaseReport, DefaultDest, GridHoriz, GridPen, OutputFileName, ReportDest, RulerType, SystemFiler, SystemOptions, SystemPreview, SystemPrinter, SystemSetups, TitlePreview, TitleSetup, TitleStatus, Version

Principais Métodos

Este componente não possui métodos associados.

Principais Eventos

OnPreviewSetup, OnPreviewShow, OverridePreview, OverrideSetup, OverrideStatus

TRvTableConnection

Unit

Na VCL:

RpConBDE

Na CLX:

QRpConBDE

Descrição

Usado para geração de relatórios vinculados a dados provenientes de um componente TTable.

Principais Propriedades

FieldAliasList, LocalFilter, RuntimeVisibility, Table, UseSetRange, Version

Principais Métodos

WriteBCDData, WriteBlobData, WriteBoolData, WriteCurrData, WriteDateTime, WriteFloatData, WriteIntData, WriteNullData, WriteStrData

Principais Eventos

OnFirst, OnGetCols, OnGetRow, OnGetSorts, OnNext, OnOpen, OnRestore, OnSetFilter, OnSetSort, OnValidateRow

TSaveDialog

Descrição

O controle TSaveDialog fornece uma caixa de diálogo para salvar arquivos.

Unit

Na VCL:

Dialogs

Na CLX:

QDialogs

Principais Propriedades

ComponentIndex, Ctl3D, DefaultExt, FileEditStyle, FileName, Files, Filter, FilterIndex, HelpContext, HelpKeyword, HelpTypeHistoryList, InitialDir, Name, Options, Owner, Tag e Title

Principais Métodos

Execute

Principais Eventos

Esse componente não possui eventos associados.

TSavePictureDialog

Descrição

O controle TSavePictureDialog fornece uma caixa de diálogo para salvar arquivos que armazenam imagens.

Unit

Na VCL :

Dialogs

Principais Propriedades

ComponentIndex, Ctl3D, DefaultExt, FileEditStyle, FileName, Files, Filter, FilterIndex, HelpContext, HelpKeyword, HelpTypeHistoryList, InitialDir, Name, Options, Owner, Tag e Title

Principais Métodos

Execute

Principais Eventos

Esse componente não possui eventos associados.

TScreen

Descrição

Esse componente representa a tela do computador no qual a aplicação está sendo executada. O Delphi sempre cria automaticamente um componente chamado Screen do tipo TScreen para a sua aplicação.

Unit

Na VCL:

Forms

Na CLX:

QForms

Principais Propriedades

ActiveControl, ActiveForm, ComponentCount, ComponentIndex, Components, Cursor, Cursors, FormCount, Forms, Fonts, Heigth, Name, Owner, PixelsPerInch, Tag e Width

Principais Métodos

FindComponent, InsertComponent e RemoveComponent

Principais Eventos

OnActiveControlChange e OnActiveFormChange

TScrollBar

Descrição

O controle TScrollBar consiste em uma barra de rolamento, usada para "rolar" o conteúdo de uma janela, formulário ou controle.

Unit

Na VCL:

StdCtrls

Na CLX:

QStdCtrls

Principais Propriedades

Align, ComponentIndex, Ctrl3D, Cursor, DragCursor, DragMode, Enabled, Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, Kind, LargeChange, Left, Max, Min, Name, Owner, Parent, ParentCtl3D, ParentShowHint, PopupMenu, Position, ShowHint, Showing, SmallChange, TabOrder, TabStop, Tag, Top, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, SendToBack, SetBounds, SetFocus, SetParams, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyDown, OnKeyPress, OnKeyUp e OnScroll

TScrollBox

Descrição

O controle TScrollBox permite que se defina uma área de rolagem retangular interna a um formulário. Você pode, por exemplo, colocar dois controles do tipo TPanel, um para ser usado como barra de ferramentas (com propriedade Align igual a alTop) e o outro como barra de status (com propriedade Align igual a alBottom) e definir uma caixa de rolagem que ocupe a área-cliente restante, de forma que o seu conteúdo possa ser rolado sem afetar a barra de ferramentas e a barra de status.

Unit

Na VCL:

Forms

Na CLX:

QForms

Principais Propriedades

Align, BorderStyle, Brush, Color, ComponentCount, ComponentIndex, Components, ControlCount, Controls, Ctrl3D, Cursor, DragCursor, DragMode, Enabled, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, HorzScrollBar, Left, Name, Owner, Parent, ParentColor, ParentCtl3d, ParentFont, ParentShowHint, PopupMenu, ShowHint, Showing, TabOrder, TabStop, Tag, Top, VertScrollBar, Visible e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, ClientToScreen, ContainsControl, Dragging, EndDrag, FindComponent, Focused, GetTextBuf, GetTextLen, Hide, InsertComponent, InsertControl, Invalidate, Refresh, RemoveComponent, Repaint, ScaleBy, ScreenToClient, ScrollBy, ScrollInView, SendToBack, SetBounds, SetFocus, SetTextBuf, Show, e Update

Principais Eventos

OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnMouseDown, OnMouseMove, OnMouseUp e OnResize

TServerSocket

Descrição

Esse componente gerencia conexões TCP/IP do lado servidor.

Unit

Na VCL:

scktcomp

Principais Propriedades

Socket, ServerType, ThreadCacheSize, Active, Port, Service

Principais Métodos

Create, Destroy, Close e Open

Principais Eventos

OnClientConnect, OnClientDisconnect, OnClientError, OnClientRead, OnClientWrite, OnGetSocket, OnGetThread, OnThreadEnd, OnThreadStart, OnAccept, OnListen

TSession

Descrição

Esse componente não pode ser explicitamente criado pelo programador, mas o Delphi sempre cria automaticamente um componente chamado Session do tipo TSession para a sua aplicação. Esse componente se encarrega de controlar a conexão aos bancos de dados relacionados com a aplicação.

Unit

Na VCL:

```
DBTables
```

Principais Propriedades

```
DataBaseCount, DataBases, Handle, KeepConnections, Locale, Name, NetFileDir, Owner, PrivateDir e Tag
```

Principais Métodos

```
AddPassword, CloseDataBase, DropConnections, FindDataBase, GetAliasNames, GetAliasParams,
GetDatabaseNames, GetDriverNames, GetDriverParams, GetPassword, GetTableNames, GetStoredProcNames,
OpenDatabase, RemoveAllPasswords e RemovePassWord
```

Principais Eventos

```
OnPassword
```

TShape

Descrição

O componente TShape permite a exibição de formas geométricas em um formulário.

Unit

Na VCL:

```
ExtCtrls
```

Na CLX:

```
QExtCtrls
```

Principais Propriedades

```
Align, BoundsRect, Brush, ComponentIndex, Cursor, DragCursor, DragMode, Enabled, Height, Hint, Left,
Name, Owner, Parent, ParentShowHint, Pen, Shape, ShowHint, Tag, Top, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, ClientToScreen, Dragging, EndDrag, Hide, Invalidate, Refresh, Repaint,
ScaleBy, ScreenToClient, SendToBack, SetBounds, Show e Update
```

Principais Eventos

```
OnDragDrop, OnDragOver, OnEndDrag, OnMouseDown, OnMouseMove e OnMouseUp
```

TSimpleDataSet

Descrição

Este componente pode ser considerado como uma reunião de três componentes: um SQLDataSet, um DatasetProvider e um ClientDataset. Substitui o componente SQLClientDataset da versão 6.

Unit

Na VCL e na CLX:

```
SimpleDS
```

Principais Propriedades

```
Active, ActiveAggs, AggFields, Aggregates, AggregatesActive, Appserver, AutoCalcFields,
BlockReadSize, Bof, Bookmark, CanModify, ChangeCount, CloneSource, CommandText, CommandType,
ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle,
```

ConnectionName, Constraints, Data, DataSetField, DataSize, DataSource, DBConnection, DefaultFields, Delta, Designer, DesignInfor, DisableStringTrim, Eof, FetchOnDemand, FieldCound, FieldDefList, FieldDefs, FieldList, Fields, FieldValues, FileName, Filter, Filtered, FilterOptions, Found, GroupingLevel, HasProvider, IndexDefs, IndexFieldCount, IndexFieldNames, IndexFields, IndexName, IsUnidirecional, KeyExclusive, KeyFieldCount, KeySize, LogChanges, MasterFields, MasterSource, Modified, Name, ObjectView, Options, Owner, PacketRecords, Params, ReadOnly, RecNo, RecordCount, RecordSize, SparseArrays, State, StatusFilter, Tag, UpdateMode, VCLComObject e XMLData

Principais Métodos

ActiveBuffer, AddIndex, AfterConstruction, Append, AppendData, AppendRecord, ApplyRange, ApplyUpdates, Assign, BookmarkValid, Cancel, CancelRange, CancelUpdates, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, ClenupInstance, ClearFields, CloneCursor, CompareBookmarks, ConstraintsDisabled, ControlsDisabled, Create, CreateBlobStream, CreateDataSet, CursorPosChanged, DataRequest, DefaultHandler, Delete, DeleteIndex, Destroy, DestroyComponents, Destroying, DisableConstraints, DisableControls, Dispatch, Edit, EditKey, EditRangeEnd, EditRangeStar, EmptyDataSet, EnableConstraints, EnableControls, Execute, ExecuteAction, FetchBlobs, FetchDetails, FetchParams, FieldAddress, FieldByName, FindComponent, FindField, FindFirst, FindKey, FindLast, FindNearest, FindNext, FindPrior, First, Free, FreeBookmarl, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDatasets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetGroupState, GetIndexInfo, GetIndexNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetNextPacket, GetOptionalParam, GetParentComponent, GetQuoteChar, GotoBookmark, GotoCurrent, GotoKey, GotoNearest, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsImplementorOf, IsLinkedTo, IsSequenced, Last, LoadFromFile, LoadFromStream, Locate, Lookup, MergeChangeLog, MethodAddress, MethodName, MoveBy, NewInstance, Next, Open, Post, Prior, Reconcile, ReferenceInterface, Refresh, RefreshRecord, RemoveComponent, RemoveFreeNotification, Resync, RevertRecord, SaveCallException, SaveToFile, SaveToStream, SetAltRecBuffers, SetFields, SetKey, SetOptionalParam, SetProvider, SetRange, SetRangeEnd, SetRangeStart, SetSubCompnent, Translate, UndoLastChange, UpdateAction, pdateCursorPos, UpdateRecord e UpdateStatus

Principais Eventos

AfterApplyUpdates, AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterExecute, AfterGetParams, AfterGetRecords, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterRowRequest, AfterScroll, AfterUpdateRecord, BeforeApplyUpdates, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeExecute, BeforeGetParams, BeforeGetRecords, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeRowRequest, BeforeScroll, BeforeUpdateRecord, OnCalcFields, OnDeleteError, OnEditError, OnFilterRecord, OnGetTableName, OnNewRecord, OnPostError, OnReconcileError, OnUpdateData e OnUpdateError

TSimpleObjectBroker

Descrição

Esse componente é responsável por fornecer uma lista de aplicações-servidoras a um componente de conexão.

Unit

Na VCL:

ObjBrkr

Principais Propriedades

LoadBalanced, ServerData, Servers

Principais Métodos

Create, Destroy, GetComputerForGUID, GetComputerForProgID, GetPortForComputer, LoadFromStream, SaveToStream, SetConnectStatus

Principais Eventos

Esse componente não possui eventos associados.

TSMALLINTFIELD

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um número inteiro, variando de -32,768 a 32,767.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsSQLTimeStamp, AsString, Calculated, CanModify,
DataSet, DataSize, DataType, DisplayFormat, DisplayLabel, DisplayName, DisplayText, DisplayWisth,
EditFormat, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, MaxValue,
MinValue, Name, Owner, Precision, ReadOnly, Required, Size, Tag, Text, Value e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate
```

TSOCKETCONNECTION

Descrição

Esse componente é responsável por conectar uma aplicação-cliente a um servidor remoto em uma aplicação multicamada usando o Windows Socket.

Unit

Na VCL:

```
Sconnect
```

Principais Propriedades

```
Address, Host, InterceptGUID, Port, AppServer, Connected, LoginPrompt, ObjectBroker, ServerGUID,
ServerName
```

Principais Métodos

```
Create, CreateTransport, GetProvider, Destroy e DoConnect
```

Principais Eventos

```
AfterConnect, AfterDisconnect, BeforeConnect, BeforeDisconnect, OnGetUserName, OnLogin
```

TSPEEDBUTTON

Descrição

O controle TSpeedButton É um botão de pressionamento especial, que permite a inclusão de um bitmap na sua face, e normalmente É usado como botão de uma barra de ferramentas.

Unit

Na VCL:

```
Buttons
```

Na CLX:

```
QButtons
```

Principais Propriedades

```
Align, AllowAllUp, BoundsRect, Caption, ComponentIndex, Cursor, Down, Enabled, Font, Glyph,
GroupIndex, Height, Hint, Layout, Left, Margin, Name, NumGlyphs, Owner, Parent, ParentFont,
ParentShowHint, ShowHint, Showing, Spacing, Tag, Top, Visible e Width
```

Principais Métodos

BeginDrag, BringToFront, CanFocus, Click, ClientToScreen, Dragging, EndDrag, GetTextBuf, GetTextLen, Hide, Invalidate, Refresh, Repaint, ScaleBy, ScreenToClient, SendToBack, SetBounds, SetTextBuf, Show e Update

Principais Eventos

OnClick, OnDblClick, OnMouseDown, OnMouseMove e OnMouseUp

TSQLConnection

Descrição

Este componente É responsável pelo acesso a bancos de dados através do Mecanismo DBExpress.

Unit

Na VCL e na CLX:

sqlExpr

Principais Propriedades

ActiveStatements,AutoClone, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, Connected, ConnectionName, ConnectionState, DataSetCount, DataSets, DesignInfo, DriverName, GetDriverFunc, InTransaction, KeepConnection, LibraryName, LoadParamsOnConnect, LocaleCode, LoginPrompt, MaxStmtsPerConn, MetaData, Name, Owner, Params, ParamsLoaded, SQLConnection, SQLHourGlass, TableScope, Tag, TraceCallbackEvent, TransactionsSupported, VendorLib

Principais Métodos

AfterConstruction, Assign, BeforeDestruction, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, CloneConnection, Close, CloseDataSets, Commit, Create, DefaultHandler, Destroy, DestroyComponents, Destroying, Dispatch, Execute, ExecuteAction, ExecuteDirect, FieldAddress, FindComponent, Free, FreeInstance, FreeNotification, GetFieldNames, GetIndexNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GetProcedureNames, GetProcedureParams, GetTableNames, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, IsImplementorOf, LoadParamsFromIniFile, MethodAddress, MethodName, NewInstance, Open, ReferenceInterface, RemoveComponent, RemoveFreeNotification, Rollback, SafeCallException, SetSubComponent, SetTraceCallbackEvent, StartTransaction, UpdateAction

Principais Eventos

AfterConnect, AfterDisconnect, BeforeConnect, BeforeDisconnect, OnLogin.

TSQLDataSet

Descrição

Este componente permite acesso direto ou via declarações sql a tabelas de bancos de dados através do Mecanismo DBExpress.

Unit

Na VCL e na CLX:

sqlExpr

Principais Propriedades

Active, AggFields, AutoCalcFields, BlockReadSize, Bof, Bookmark, CanModify, CommandText, CommandType, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DataSetField, Datasource, DefaultFields, Designer, DesignerData, DesignInfo, Eof, FieldCount, FieldDefList, FieldDefs, FieldList, Fields, FieldValues, Filter, Filtered, FilterOptions, Found, IndexFields, IsUnidirecional, MaxBlobSize, Modified, Name, ObjectView, Owner, ParamCheck, Params, Prepared, RecNo, RecordCount, RecordSize, SortFieldNames, SparseArrays, SQLConnection, State, Tag, TransactionLevel.

Principais Métodos

ActiveBuffer, AfterConstruction, Append, AppendRecord, Assign, BeforeDestruction, BookmarkValid, Cancel, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CursorPosChanged, DefaultHandler, Delete,

Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EnableControls, ExecSQL, ExecuteAction, FieldAddress, FieldByName, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GotoBookmark, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsImplementorOf, IsLinkedTo, IsSequenced. Last, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, Open, ParamByName, Post, Prior, ReferenceInatance, Refresh, RemoveComponent, Resync, SafeCallException, SetFields, SetSchemaInfo, SetSubComponent, Translate, UpdateAction, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnFilterRecord, OnNewRecord, OnPostError

TSQLMONITOR

Descrição

Este componente Permite interceptar comandos sql entre componentes de acesso e o serevidor de banco de dados, através do Mecanismo DBExpress.

Unit

Na VCL e na CLX:

sqlExpr

Principais Propriedades

Active, AutoSave, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle,DesignInfo, FileName, MaxTraceCount, Name, Owner, SQLConnection, Tag, TraceCount, TraceList e VCLComObject.

Principais Métodos

AfterConstruction, Assign, BeforeDestruction, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, Create, Destroy, DestroyComponents, Destroying, Dispatch, ExecuteAction, FieldAddress, FindComponent, Free, FreeInstance, FreeNotification, FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, IsImplementorOf, LoadFromFile, MethodAddress, MethodName, NewInstance, ReferenceInterface, RemoveComponent, RemoveFreeNotification, SafeCallException, SaveToFile, SetSubComponent e UpdateAction

Principais Eventos

OnLogTrace e OnTrace

TSQLQUERY

Descrição

Este componente permite acesso via declarações sql a tabelas de bancos de dados através do Mecanismo DBExpress.

Unit

Na VCL e na CLX:

sqlExpr

Principais Propriedades

Active, AggFields, AutoCalcFields, BlockReadSize, Bof, Bookmark, BookmarkSize, BufferCount, Buffers, CacheSize, CalcBuffer, CalcFieldsSize, CanModify, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DataSetField, DataSource, DefaultFields, Designer, DesignerData, DesignInfo, Eof, FieldCount, FieldDefList, FieldDefs, FieldList, Fields, FieldValues, Filter, Filtered, FilterOptions, Found, IndexFields, IsUnidirecional, MaxBlobSize, Modified, Name, NoMetadata, ObjectView, Owner, ParamCheck, Params, Prepared, RecNo, RecordCount, RecordSize, RowsAffected, SparseArrays, SQL, SQLConnection, State, Tag, Text, TransisolationLevel, VCLComObject,

Principais Métodos

ActiveBuffer, AfterConstruction, Append, AppendRecord, Assign, BeforeDestruction, BookmarkValid, Cancel, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CursorPosChanged, DefaultHandler, Delete, DeleteRecords, Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EnableControls, ExecSQL, ExecuteAction, FieldAddress, FieldByName, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GETKEYFIELDNAMES, GetNamePath, GetParentComponent, GetQuoteChar, GotoBookmark, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsImplementorOf, IsLinkedTo, IsSequenced. Last, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, Open, ParamByName, Post, PrepareStatement, Prior, ReferenceInterface, Refresh, RemoveComponent, RemoveFreeNotification, Resync, SafeCallException, SetFields, SetSchemaInfo, SetSubComponent, Translate, UpdateAction, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnFilterRecord, OnNewRecord, OnPostError

TSQLStoredProc

Descrição

Esse componente permite que uma aplicação desenvolvida em Delphi execute procedimentos armazenados em servidores, através do mecanismo DBExpress.

Unit

Na VCL e na CLX:

sqlExpr

Principais Propriedades

Active, AggFields, AutoCalcFields, BlockReadSize, Bof, Bookmark, CanModify, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DataSetField, DataSource, DefaultFields, Designer, DesignerData, DesignInfo, Eof, FieldCount, FieldDefList, FieldDefs, FieldList, Fields, FieldValues, Filter, Filtered, FilterOptions, Found, IndexDefs, IsUnidirecional, MaxBlobSize, Modified, Name, ObjectView, Owner, ParamCheck, Params, Prepared, RecNo, RecordCount, RecordSize, SparseArrays, SQLConnection, State, Tag, VCLComObject

Principais Métodos

ActiveBuffer, AfterConstruction, Append, AppendRecord, Assign, BeforeDestruction, BookmarkValid, Cancel, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, ClearFields, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CursorPosChanged, DefaultHandler, Delete, Destroy, DestroyComponents, Destroying, DisableControls, Dispatch, Edit, EnableControls, ExecProc, ExecuteAction, FieldAddress, FieldByName, FindComponent, FindField, FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification, FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets, GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetKeyFieldNames, GetNamePath, GetParentComponent, GotoBookmark, HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty, IsImplementorOf, IsLinkedTo, IsSequenced. Last, Locate, Lookup, MethodAddress, MethodName, MoveBy, NewInstance, Next, NextRecordset, Open, ParamByName, Post, PrepareStatement, Prior, ReferenceInstance, Refresh, RemoveComponent, RemoveFreeNotification, Resync, SafeCallException, SetSchemaInfo, SetSubComponent, Translate, UpdateAction, UpdateCursorPos, UpdateRecord, UpdateStatus

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh, AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnFilterRecord, OnNewRecord, OnPostError

TSQLTABLE

Descrição

Este componente permite acesso direto a tabelas de bancos de dados através do Mecanismo Activex Data Objects (ADO).

Unit

Na VCL e na CLX:

```
sqlExpr
```

Principais Propriedades

```
Active, AggFields, AutoCalcFields, BlockReadSize, Bof, Bookmark, CanModify, ComObject,
ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DataSetField,
DataSource, DefaultFields, Designer, DesignerData, DesignInfo, Eof, FieldCount, FieldDefList,
FieldDefs, FieldList, Fields, FieldValues, Filter, Filtered, FilterOptions, Found, IndexFields,
IndexName, IsUnidirecional, MasterFields, Mastersource, MaxBlobSize, Modified, Name, NoMetadata,
ObjectView, Owner, Prepared, RecNo, RecordCount, RecordSize, SparseArrays, SQL, SQLConnection,
State, TableName, Tag, TransisolationLevel, VCLComObject,
```

Principais Métodos

```
ActiveBuffer, AfterConstruction, Append, AppendRecord, Assign, BeforeDestruction, BookmarkValid,
Cancel, CheckBrowseMode, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance,
ClearFields, Close, CompareBookmarks, ControlsDisabled, Create, CreateBlobStream, CursorPosChanged,
DefaultHandler, Delete, DeleteRecords, Destroy, DestroyComponents, Destroying, DisableControls,
Dispatch, Edit, EnableControls, ExecuteAction, FieldAddress, FieldByName, FindComponent, FindField,
FindFirst, FindLast, FindNext, FindPrior, First, Free, FreeBookmark, FreeInstance, FreeNotification,
FreeOnRelease, GetBlobFieldData, GetBookmark, GetCurrentRecord, GetDetailDataSets,
GetDetailLinkFields, GetFieldData, GetFieldList, GetFieldNames, GetInterface, GetInterfaceEntry,
GetInterfaceTable, GetKeyFieldNames, GetNamePath, GetParentComponent, GetQuoteChar, GotoBookmark,
HasParent, InheritsFrom, InitInstance, Insert, InsertComponent, InsertRecord, InstanceSize, IsEmpty,
IsImplementorOf, IsLinkedTo, IsSequenced. Last, Locate, Lookup, MethodAddress, MethodName, MoveBy,
NewInstance, Next, Open, ParamByName, Post, PrepareStatement, Prior, ReferenceInterface, Refresh,
RemoveComponent, RemoveFreeNotification, Resync, SafeCallException, SetFields, SetSchemaInfo,
SetSubComponent, Translate, UpdateAction, UpdateCursorPos, UpdateRecord, UpdateStatus
```

Principais Eventos

```
AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, AfterRefresh,
AfterScroll, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen,
BeforePost, BeforeRefresh, BeforeScroll, OnCalcFields, OnDeleteError, OnEditError, OnFilterRecord,
OnNewRecord, OnPostError.
```

TSTACK

Descrição

Esta classe permite manipular objetos usando uma estrutura de dados do tipo Pilha. Neste tipo de estrutura de dados, o primeiro elemento a entrar é o último a sair, sendo comumente denominada estrutura do tipo LIFO – "Last In, First Out"

Unit

```
Contnrs
```

Principais Propriedades

```
Esta classe não possui propriedades associadas.
```

Principais Métodos

```
AfterConstruction, AtLeast, BeforeDestruction, ClassInfo, ClassName, ClassNameIs, ClassParent,
ClassType, CleanupInstance, Count, Create, DefaultHandler, Destroy, Dispatch, FieldAddress, Free,
FreeInstance, GetInterface, GetInterfaceEntry, GetInterfaceTable, InheritsFrom, InitInstance,
InstanceSize, MethodAddress, MethodName, NewInstance, Peek, Pop, Push, SafeCallException
```

Principais Eventos

Esta classe não possui eventos associados.

TStatusBar

Descrição

O controle TStatusBar é um componente que permite que se exiba uma barra de status na janela principal de uma aplicação (geralmente a barra de status é horizontal e fica situada na parte inferior da janela) para transmitir informações ao usuário.

Unit

Na VCL:

```
ComCtrls
```

Na CLX:

```
QComCtrls
```

Principais Propriedades

```
Align, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect, ClientWidth, ComponentCount,
ComponentIndex, Components, ControlCount, Controls, Cursor, DragCursor, DragMode, Enabled, Font,
Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Owner, Panels, Parent,
ParentFont, ParentShowHint, PopupMenu, ShowHint, SimplePanel, SimpleText, SizeGrip, Tag, Top,
Visible e Width
```

Principais Métodos

```
Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen,
ContainsControl, ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindComponent, Focused,
Free, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl,
Invalidate, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, ScrollBy,
SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnClick, OnDblClick, OnDragDrop, OnDragOver, OnDrawPanel, OnEndDrag, OnMouseDown, OnMouseMove,
OnMouseUp, OnResize e OnStartDrag
```

TStoredProc

Descrição

Esse componente permite que o Delphi execute procedimentos armazenados em servidores.

Unit

Na VCL:

```
DBTables
```

Principais Propriedades

```
Active, AutoCalcFilelds, BOB, CanModify, Database, DatabaseName, DBHandle, DBLocale, EOF,
FieldCount, FieldDefs, Field, Handle, Locale, Modified, Name, Owner, Overload, ParamBindMode,
ParamCount, Params, Prepared, RecordCount, State, StmtHandle, StoredProcName e Tag
```

Principais Métodos

```
Append, AppendRecord, Cancel, CheckBrowseMode, ClearFields, Close, CopyParams, CursorPosChanged,
Delete, DescriçãosAvailable, DisableControls, Edit, EnableControls, ExecProc, FieldByName,
FindField, First, FreeBookmark, GetBookmark, GetFieldNames, GetResults, GotoBookmark, Insert,
InsertRecord, Last, MoveBy, Next, Open, ParamByName, Post, Prepare, Prior, Refresh, SetFields,
UnPrepared, UpdateCursorPos e UpdateRecord
```

Principais Eventos

```
AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, BeforeCancel,
BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, OnCalcFields e
OnNewRecord
```

TStringField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena uma informação de texto com até 255 caracteres (string).

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, ASSQLTimeStamp, AsString, Calculated, CanModify,
DataSet, DataSize, DataType, DisplayLabel, DisplayName, DisplayText, DisplayWisth, EditMask,
EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, Name, Owner, ReadOnly, Required, Size,
Tag, Text, Transliterate, Value e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate
```

TStringGrid

Descrição

O controle TStringGrid permite a exibição de um conjunto de strings e dados na forma de um arranjo de linhas e colunas.

Unit

Na VCL:

```
Grids
```

Na CLX:

```
QGrids
```

Principais Propriedades

```
Align, BorderStyle, BoundsRect, Canvas, Cells, ClientHeigth, ClentOrigin, ClientRect, ClientWidth,
Col, ColCount, Color, Cols, ColWidthds, ComponentCount, ComponentIndex, Components, ControlCount,
Controls, DefaultRow, DockClirentCount, DockClients, DragCursor, DragMode, EditorMode, Enabled,
FixedColors, FixedCols, FixedRows, Font, GridHeight, GridLineWidth, GridWidth, Handle, Height,
HelpContext, HelpKeyword, HelpTypeHint, Left, LeftCol, Name, ParentColor, ParentCtrl3D, ParentFont,
ParentShowHint, PopupMenu, Row, RowCount, RowHeights, Rows, ScrollBars, Selection, ShowHint,
Showing, TabOrder, TabStop, TabStops, Tag, Top, TopRow
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, CellRect, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf,
GetTextLen, Hide, Invalidate, MouseToCell, Refresh, RemoveComponent, Repaint, ScaleBy,
ScreenToClient, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnClick, OnColumnMoved, OnDblClick, OnDragDrop, OnDragOver, OnDrawCell, OnEndDrag, OnEnter, OnExit,
OnGetEditMask, OnGetEditText, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp,
OnRowMoved, OnSelectCell, OnSetEditText e OnTopLeftChanged
```

TStringList

Descrição

Esse objeto é utilizado para manter uma lista de strings.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
Count, Duplicates, Objects, Sorted, Strings e Values
```

Principais Métodos

```
Add, AddObject, AddStrings, Assign, BeginUpdate, ClassName, ClassParent, ClassType, Clear, Delete,
EndUpdate, Exchange, Free, GetText, IndexOf, IndexOfObject, Insert, InsertObject, LoadFromFile,
SaveToFile, SetText e Sort
```

Principais Eventos

```
OnChange
```

TStrings

Descrição

Esse objeto é utilizado para manipular strings.

Unit

Na VCL e na CLX:

```
Classes
```

Principais Propriedades

```
Count, Objects, Strings e Values
```

Principais Métodos

```
Add, AddObject, AddStrings, Assign, BeginUpdate, ClassName, ClassParent, ClassType, Clear, Delete,
EndUpdate, Exchange, Free, GetText, IndexOf, IndexOfObject, Insert, InsertObject, LoadFromFile,
Move, SaveToFile, SetText e Sort
```

Principais Eventos

```
OnChange
```

TTabbedNotebook

Descrição

O componente TTabbedNotebook é um componente que possui muitas páginas, cada uma com os seus próprios controles. O usuário pode selecionar uma página clicando com o mouse sobre a guia correspondente.

Unit

Na VCL:

```
TabNotBk
```

Principais Propriedades

```
ActivePage, Align, BoundsRect, ComponentIndex, ControlCount, Controls, Cursor, Enabled, Font,
Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Owner, PageIndex, Pages, Parent,
TabOrder, TabsPerRow, TabStop, Tag, Top, Visible e Width
```

Principais Métodos

```
BeginDrag, BringToFront, CanFocus, ClientToScreen, ContainsControl, Dragging, EndDrag,
FindComponent, Focused, GetIndexForPage, GetTextBuf, GetTextLen, Hide, InsertControl, Invalidate,
Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTabFocus,
SetTextBuf, Show e Update
```

Principais Eventos

```
OnEnter e OnExit
```

TTABCONTROL

Descrição

Esse controle é semelhante ao controle TTabSet, e fornece um conjunto de guias horizontais que o usuário pode selecionar para iniciar determinadas ações. Normalmente, esse controle é usado em conjunto com um controle do tipo TPageControl.

Unit

Na VCL:

```
ComCtrls
```

Na CLX:

```
QComCtrls
```

Principais Propriedades

```
Align, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect, ClientWidth, ComponentCount,
ComponentIndex, Components, ControlCount, Controls, Cursor, DesignInfo, DisplayRect, DragCursor,
DragMode, Enabled, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, MultiLine,
Name, Owner, Parent, ParentFont, ParentShowHint, PopupMenu, ShowHint, TabHeight, TabIndex, TabOrder,
TabStop, TabWidth, Tabs, Tag, Top, Visible e Width
```

Principais Métodos

```
Assign, BeginDrag, BringToFront, BroadCast, CanFocus, ClassInfo, ClassName, ClassNamels,
ClassParent, ClassType, CleanupInstance, ClientToScreen, ContainsControl, ControlAtPos, Create,
Destroy, DestroyComponents, Destroying, DisableAlign, Dispatch, DragDrop, Dragging, EnableAlign,
EndDrag, FieldAddress, FindComponent, Focused, Free, FreeInstance, FreeNotification,
GetTabOrderList, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InheritsFrom,
InitInstance, InsertComponent, InsertControl, InstanceSize, Invalidate, MethodAddress, MethodName,
NewInstance, PaintTo, Perform, Realign, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy,
ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show, Update e
UpdateControlState
```

Principais Eventos

```
OnChange, OnChanging, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnMouseDown, OnMouseMove,
OnMouseUp, OnStartDrag
```

TTABLE

Descrição

Esse componente permite acesso a tabelas de bancos de dados por meio do Borland Database Engine.

Unit

Na VCL:

```
DBTables
```

Principais Propriedades

```
Active, AutoCalcFilelds, BOB, CanModify, Database, DatabaseName, DBHandle, DBLocale, DatabaseSource,
DBLocale, EOF, Exclusive, FieldCount, FieldDefs, Fields, Handle, IndexDefs, IndexFieldCount,
IndexFieldNames, IndexName, IndexFields, KeyExclusive, KeyFieldCount, Local, MasterField,
MasterSource, Locale, Modified, Name, Owner, ReadOnly, RecordCount, State, StoredDefs, TableLevel,
TableName, TableType, Tag e UpdateMode
```

Principais Métodos

```
AddIndex, ApplyRange, Append, AppendRecord, BatchMove, Cancel, CancelRange, CheckBrowseMode,
ClearFields, Close, CreateTable, CursorPosChanged, Delete, DeleteIndex, DeleteTable,
DisableControls, Edit, EditKey, EditRangeEnd, EditRangeStart, EmptyTable, EnableControls,
FieldByName, FindField, FindKey, FindNearest, First, FreeBookmark, GetBookmark, GetFieldNames,
GetIndexNames, GotoBookmark, GotoCurrent, GotoKey, GotoNearest, Insert, InsertRecord, Last, MoveBy,
Next, Open, Post, Prior, Refresh, SetFields, SetKey, SetRange, SetRamgeEnd, SetRamgeStart e
UpdateRecord
```

Principais Eventos

AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterInsert, AfterOpen, AfterPost, BeforeCancel, BeforeClose, BeforeDelete, BeforeEdit, BeforeInsert, BeforeOpen, BeforePost, OnCalcFields e OnNewRecord

TTabSet

Descrição

O componente TTabSet fornece um conjunto de guias horizontais que o usuário pode selecionar para iniciar determinadas ações. Normalmente, esse controle é usado em conjunto com um controle do tipo TNotebook.

Unit

Na VCL:

Tabs

Principais Propriedades

Align, AutoScroll, BackgroundColor, Canvas, ComponentIndex, Cursor, DitherBackground, DragMode, Enabled, EndMargin, FirstIndex, Font, Height, HelpContext, HelpKeyword, HelpTypeHint, Left, Name, Owner, Parent, ParentShowHint, SelectedColor, ShowHint, Showing, StartMargin, Style, TabHeigth, TabIndex, Tabs, Tag, Top, UnselectedColor, Visible, VisibleTabs e Width

Principais Métodos

BeginDrag, BringToFront, CanFocus, ClientToScreen, Dragging, EndDrag, Focused, GetTextBuf, GetTextLen, Hide, Invalidate, ItemAtPos, ItemRect, Refresh, Repaint, ScaleBy, ScreenToClient, ScrollBy, SelectNext, SendToBack, SetBounds, SetFocus, SetTextBuf e Update

Principais Eventos

OnChange, OnClick, OnDragDrop, OnDragOver, OnDrawTab, OnEndDrag, OnEnter, OnExit e OnMeasureItem

TTabSheet

Descrição

Esse componente representa uma página de um controle do tipo TPageControl.

Unit

Na VCL:

ComCtrls

Na CLX:

QComCtrls

Principais Propriedades

Align, BoundsRect, Brush, Caption, ClientHeight, ClientOrigin, ClentRect, ClientWidth, ComponentCount, ComponentIndex, Components, ControlCount, Controls, Cursor, Enabled, Handle, Height, HelpContext, HelpKeyword, HelpTypeLeft, Name, Owner, PageControl, PageIndex, Parent, ParentShowHint, PopupMenu, ShowHint, TabIndex, TabOrder, TabStop, TabVisible, Tag, Top, Visible e Width

Principais Métodos

Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen, ContainsControl, ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindComponent, Focused, Free, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl, Invalidate, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnEnter e OnExit

TThread

Descrição

Essa classe permite desenvolver aplicações com vários processos independentes (cada processo é denominado um thread ou canal de execução).

Unit

Na VCL e na CLX:

```
classes
```

Principais Propriedades

```
FreeOnTerminate, Handle, Priority, ReturnValue, Suspended, Terminated, ThreadID
```

Principais Métodos

```
Create, Destroy, DoTerminate, Execute, Resume, Suspend, Synchronize, Terminate e WaitFor
```

Principais Eventos

```
OnTerminate
```

TTimeField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena uma informação de uma hora.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsSQLTimeStamp, AsString, Calculated, CanModify,
DataSet, DataSize, DataType, DisplayLabel, DisplayName, DisplayText, DisplayWidth, EditMask,
EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, Name, Owner, ReadOnly, Required, Size,
Tag, Text, e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate
```

TTimer

Descrição

Esse componente dispara um evento OnTimer a intervalos regulares de tempo.

Unit

Na VCL:

```
ExtCtrls
```

Na CLX:

```
QExtCtrls
```

Principais Propriedades

```
ComponentIndex, Enabled, Interval, Name, Owner e Tag
```

Principais Métodos

Esse componente não possui métodos associados.

Principais Eventos

OnTimer

TToolBar

Descrição

O componente TToolBar é utilizado para criação de barras de ferramentas no padrão Windows.

Unit

Na VCL:

ComCtrls

Na CLX:

QComCtrls

Principais Propriedades

ButtonCount, ButtonHeight, Buttons, ButtonWidth, Constraints, DisabledImages, DockSite, DragKind, Flat, HotImages, Images, Indent, List, RowCount, ShowCaptions, Transparent, Wrapable, BorderWidth, EdgeBorders, EdgeInner, EdgeOuter, Brush, ClientOrigin, ClientRect, ControlCount, Controls, Ctl3D, Handle, HelpContext, HelpKeyword, HelpTypeParentWindow, Showing, TabOrder, TabStop, Align, Anchors, AutoSize, BoundsRect, Caption, ClientHeight, ClientWidth, Color, ControlState, ControlStyle, Cursor, DragCursor, DragMode, Enabled, Font, Height, Hint, Left, Name, Parent, ParentColor, ParentFont, ParentShowHint, PopupMenu, ShowHint, Top, Visible, Width, WindowProc, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DesignInfo, Owner, Tag, VCLComObject

Principais Métodos

CanAutoSize, CancelMenu, ChangeScale, CheckMenuDropdown, ClickButton, Create, Destroy, FindButtonFromAccel, InitMenu, TrackMenu, WrapButtons

Principais Eventos

OnDockDrop, OnDockOver, OnEnter, OnExit, OnGetSiteInfo, OnUnDock, OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDock, OnEndDrag, OnMouseDown, OnMouseMove, OnMouseUp, OnResize, OnStartDock, OnStartDrag

TTrackBar

Descrição

O controle TTrackBar consiste em uma barra opcionalmente graduada, usada para definir valores em uma escala e possuindo um controle deslizante que permite alterar o valor corrente.

Unit

Na VCL:

ComCtrls

Na CLX:

QComCtrls

Principais Propriedades

Align, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect, ClientWidth, ComponentCount, ComponentIndex, Components, ControlCount, Controls, Ctl3D, Cursor, DragCursor, DragMode, Enabled, Frequency, Handle, Height, Hint, HelpContext, HelpKeyword, HelpTypeLeft, LineSize, Max, Min, Name, Orientation, Owner, PageSize, Parent, ParentCtl3D, ParentShowHint, PopupMenu, Position, ReadOnly, SelEnd, SelStart, ShowHint, Showing, TabOrder, TabStop, Tag, TickMarks, TickStyle, Top, Visible e Width

Principais Métodos

Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen, ContainsControl, ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindComponent, Focused, Free, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl, Invalidate, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnDragDrop, OnDragOver, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp e OnStartDrag

TTREEVIEW

Descrição

O controle TTreeView é um componente que permite a exibição de itens em vários níveis hierárquicos, nos quais cada item é composto de um rótulo e de um número variável de bitmaps (opcional).

Unit

Na VCL:

ComCtrls

Na CLX:

QComCtrls

Principais Propriedades

Align, BorderStyle, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect, ClientWidth, Color, ComponentCount, ComponentIndex, Components, ControlCount, Controls, Ctl3D, Cursor, DragCursor, DragMode, DropTarget, Enabled, Font, Handle, Height, HelpContext, HelpKeyword, HelpTypeHideSelection, Hint, Images, Indent, Items, Left, Name, Owner, Parent, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PopupMenu, ReadOnly, Selected, ShareImages, ShowButtons, ShowHint, Showing, ShowLines, ShowRoot, Sorted, SortType, StateImages, TabOrder, TabStop, Tag, Top, TopItem, Visible e Width

Principais Métodos

AlphaSort, Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen, ContainsControl, ControlAtPos, Create, CustomSort, Destroy, DragDrop, Dragging, EndDrag, FindComponent, Focused, Free, FullCollapse, FullExpand, GetHitTestInfoAt, GetNodeAt, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl, Invalidate, IsEditing, LoadFromFile, LoadFromStream, Refresh, RemoveComponent, RemoveControl, Repaint, SaveToFile, SaveToStream, ScaleBy, ScreenToClient, ScrollBy, SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update

Principais Eventos

OnChange, OnChanging, OnClick, OnCollapsed, OnCollapsing, OnCompare, OnDblClick, OnDeletion, OnDragDrop, OnDragOver, OnEdited, OnEditing, OnEndDrag, OnEnter, OnExit, OnExpanded, OnExpanding, OnGetImageIndex, OnGetSelectedIndex, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseDown, OnMouseMove, OnMouseUp e OnStartDrag

TUPDATESQL

Descrição

O componente TUpdateSQL fornece um meio alternativo de se atualizar bancos de dados, através do mecanismo do Borland Database Engine, armazenando um comando para cada tipo de declaração SQL.

Unit

Na VCL:

DBTables

Principais Propriedades

ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle, DataSet, DeleteSQL, DesignInfo, InsertSQL, ModifySQL, Name, Owner, Query, SQL, Tag, VCLComObject

Principais Métodos

AfterConstruction, Apply, Assign, BeforeDestruction, ClassInfo, ClassName, ClassNameIs, ClassParent, ClassType, CleanupInstance, Create, DefaultHandler, Destroy, DestroyComponents, Destroying, Dispatch, ExecSQL, ExecuteAction, FieldAddress, FindComponent, Free, FreeInstance, FreeNotification, FreeOnRelease, GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, HasParent, InheritsFrom, InitInstance, InsertComponent, InstanceSize, MethodAddress, MethodName, NewInstance, RemoveComponent, SafeCallException, SetParams, UpdateAction

Principais Eventos

Esse componente não possui eventos associados.

TUpDown

Descrição

O controle TUpDown consiste em um par de botões de seta, uma apontando para cima ("Up") e outra apontando para baixo ("Down"), que, ao serem selecionadas com o botão esquerdo do mouse, provocam o incremento ou decremento de um valor numérico, armazenado na sua propriedade Position. Normalmente esse controle é usado em conjunto com outro controle (geralmente TEdit), que exibe o valor numérico armazenado na propriedade Position.

Unit

Na VCL:

```
ComCtrls
```

Principais Propriedades

```
Align, AlignButton, Associate, ArrowKeys, BoundsRect, Brush, ClientHeight, ClientOrigin, ClentRect,
ClientWidth, ComponentCount, ComponentIndex, Components, ControlCount, Controls, Cursor, Enabled,
Handle, Height, Hint, HelpContext, HelpKeyword, HelpTypeIncrement, Left, Max, Min, Name,
Orientation, Owner, Parent, ParentShowHint, PopupMenu, Position, ShowHint, Showing, TabOrder,
TabStop, Tag, Thousands, Top, Visible, Width e Wrap
```

Principais Métodos

```
Assign, BeginDrag, BringToFront, CanFocus, ClassName, ClassParent, ClassType, ClientToScreen,
ContainsControl, ControlAtPos, Create, Destroy, DragDrop, Dragging, EndDrag, FindComponent, Focused,
Free, GetTextBuf, GetTextLen, HandleAllocated, HandleNeeded, Hide, InsertComponent, InsertControl,
Invalidate, Refresh, RemoveComponent, RemoveControl, Repaint, ScaleBy, ScreenToClient, ScrollBy,
SendToBack, SetBounds, SetFocus, SetTextBuf, Show e Update
```

Principais Eventos

```
OnChanging, OnClick, OnEnter, OnExit, OnMouseDown, OnMouseMove e OnMouseUp
```

TVarBytesField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um conjunto arbitrário de até 65535 bytes.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsSQLTimeStamp, AsString, Calculated, CanModify,
DataSet, DataSize, DataType, DisplayLabel, DisplayName, DisplayText, DisplayWidth, EditMask,
EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, Name, Owner, ReadOnly, Required, Size,
Tag, Text e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate
```

TWebDispatcher

Descrição

Esse componente gerencia resposta a mensagens HTTP.

Unit

Na VCL e na CLX:

```
httpapp
```

Principais Propriedades

```
Action, Actions, Request, Response, DesignOffset, DesignSize, OldCreateOrder
```

Principais Métodos

```
ActionByName, Create, Destroy, AfterConstruction, BeforeDestruction, CreateNew
```

Principais Eventos

```
AfterDispatch, BeforeDispatch, OnCreate, OnDestroy
```

TWordField

Descrição

Esse componente representa um campo de um registro de um banco de dados que armazena um número inteiro sem sinal, variando de 0 a 65535.

Unit

Na VCL e na CLX:

```
DB
```

Principais Propriedades

```
Align, AsBoolean, AsDateTime, AsFloat, AsInteger, AsSQLTimeStamp, AsString, Calculated, CanModify,
DataSet, DataSize, DataType, DisplayFormat, DisplayLabel, DisplayName, DisplayText, DisplayWisth,
EditFormat, EditMask, EditMaskPtr, FieldName, FieldNo, Index, IsIndexField, IsNull, MaxValue,
MinValue, Name, Owner, Precision, ReadOnly, Required, Size, Tag, Text, Value e Visible
```

Principais Métodos

```
Assign, AssignValue, Clear, FocusControl, GetData, IsValidChar e SetData
```

Principais Eventos

```
OnChange, OnGetText, OnSetText e OnValidate.
```

TXMLBroker

Descrição

Este componente Permite a Manipulação de Pacotes de dados no formato XML fornecidos por uma aplicação Servidora.

Unit

```
xmlbrokr
```

Principais Propriedades

```
AppServer, ComObject, ComponentCount, ComponentIndex, Components, ComponentState, ComponentStyle,
Connected, DesignInfo, HasAppServer, MaxErrors, MaxRecords, Name, Notify, NotifyCount, Owner,
Params, ProviderName, ReconcileProducer, RemoteServer, Tag, VCLComObject, WebDispatch
```

Principais Métodos

```
AddNotify, AfterConstruction, ApplyXMLUpdates, Assign, BeforeDestruction, ClassInfo, ClassName,
ClassNameIs, ClassParent, ClassType, CleanupInstance, Create, DefaultHandler, Destroy,
DestroyComponents, Destroying, Dispatch, ExecuteAction, FetchParams, FieldAddress, FindComponent,
Free, FreeInstance, FreeNotification, FreeOnRelease, GetDelta, GetErrorCount, GetErrors,
GetInterface, GetInterfaceEntry, GetInterfaceTable, GetNamePath, GetParentComponent, GetXMLRecords,
```

HasParent, HTMLSubmitFormName, InheritsFrom, InitInstance, InsertComponent, InstanceSize, MasterRowSetVarName, MethodAddress, MethodName, NewInstance, RemoveComponent, RemoveNotify, RequestRecords, RequestUpdate, RowSetVarName, SafeCallException, SetProvider, SubmitFormVarName UpdateAction

Principais Eventos

AfterDispatch, BeforeDispatch, OnGetErrorResponse, OnGetResponse, OnRequestRecords e OnRequestUpdate

Capítulo 41

Propriedades

Aborted

Descrição

A propriedade Aborted é declarada como uma variável do tipo booleana, e indica se o usuário interrompeu o trabalho de impressão corrente. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TPrinter, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter.
```

AbortOnKeyViol

Descrição

Essa propriedade é definida como uma variável booleana que determina se a execução do Método Execute deve ser interrompida caso ocorra uma violação de integridade durante uma operação sobre um grupo de registros ou uma tabela.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante uma linha de código como:

```
BatchMove1.AbortOnKeyViol := False;
```

onde BatchMove1 é um componente do tipo TBatchMove.

Componentes aos quais se aplica:

Na fase de projeto:

```
TBatchMove
```

Durante a execução do aplicativo:

```
TBatchMove
```

AbortOnProblem

Descrição

Essa propriedade é definida como uma variável booleana que determina se a execução do Método Execute deve ser interrompida caso ocorra uma violação de integridade durante uma operação sobre um grupo de registros ou uma tabela, descartando a movimentação de dados realizada.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector, ou mediante uma linha de código como:

```
BatchMove1.AbortOnProblem := False;
```

onde BatchMove1 é um componente do tipo TBatchMove.

Componentes aos quais se aplica:

Na fase de projeto:

```
TBatchMove
```

Durante a execução do aplicativo:

```
TBatchMove
```

AccuracyMethod

Descrição

Esta propriedade é declarada como uma variável do tipo TAccuracyMethod, e indica como o texto será escrito no arquivo de projeto. Se seu valor for igual a amPositioning o texto será escrito de forma a ser reproduzido tão precisa à quanto possível na tela ou em qualquer impressora. Se seu valor for igual a amAppearance o texto não será ajustado da melhor forma possível, de forma a ter a mesma aparência na tela e na impressora.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TPrinter, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

ActionCount

Descrição

Essa propriedade é definida como uma variável inteira, e retorna o número de ações (representadas por objetos da classe TAction) definidas no componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TActionList
```

Durante a execução do aplicativo:

```
TActionList
```

ActionList

Descrição

Essa propriedade é definida como um objeto da classe TCustomActionList, e define o componente TAction ao qual o objeto está vinculado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TAction
```

Durante a execução do aplicativo:

```
TAction
```

Actions

Descrição

Essa propriedade é definida como uma array de objetos da classe TAction, e referencia as ações definidas no componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TActionList
```

Durante a execução do aplicativo:

```
TActionList
```

Active

Descrição

A propriedade Active é definida como uma variável booleana que diz se um controle está ou não ativo. Essa propriedade só está disponível durante a execução do aplicativo.

Para componentes do tipo TForm, essa propriedade não pode ter o seu valor diretamente alterado pelo usuário.

Para componentes do tipo TOLEContainer essa propriedade determina se o objeto OLE contido no componente está ou não ativo.

Para componentes dos tipos TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataSet, TIBDataset, TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery, TdecisionQuery, TStoredProc, TSQLDataset, TSQLQuery, TSQLStoredProc, TSQLTable e TSQLClientDataSet, essa propriedade determina se o banco de dados associado está ativo (aberto). Mudar o valor da propriedade para True ou False equivale a chamar os métodos Open e Close, respectivamente.

Para componentes do tipo TAnimate, indica se o clipe AVI definido na sua propriedade FileName está sendo executado.

Para objetos da classe TChartSeries, indica se a série representada pelo componente deve ou não ser exibida.

Para componentes do tipo TRvProject, indica se o projeto de relatório está ou não ativo.

Exemplo

Você pode alterar o valor da propriedade Active do componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
OLEContainer1.Active := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TAnimate, TChartSeries, TClientDataSet, TADODataset, TADOQuery, TADOStoredProc, TADOTable,
TClientDataSet, TIBDataset, TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TRvProject, TTable,
TQuery, TDecisionQuery e TStoredProc
```

Durante a execução do aplicativo:

```
TAnimate, TChartSeries, TForm, TOLEContainer, TClientDataSet, TADODataset, TADOQuery,
TADOStoredProc, TADOTable,  TClientDataSet, TIBDataset, TIBQuery, TIBStoredProc, TIBTable,
TIBTransaction, TRvProject, TTable, TQuery, TDecisionQuery e TStoredProc
```

ActiveControl

Descrição

A propriedade Active é definida como uma variável do tipo TWinControl, cujo significado depende do componente ao qual se aplica.

Para componentes do tipo TForm, essa propriedade indica o componente que possui o foco da aplicação.

Para componentes do tipo TScreen, essa propriedade indica o componente que tem o foco da aplicação, mas só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Você pode alterar a propriedade ActiveControl do componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
Form1.ActiveControl := Button1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TForm e TScreen
```

ActiveForm

Descrição

A propriedade ActiveForm é definida como uma variável do tipo TForm e indica o formulário corrente ou que receberá o foco quando a aplicação estiver ativa (no caso de o Windows estar executando uma outra aplicação). Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir muda a cor do formulário corrente para amarelo quando recebe um clique do mouse:

```
procedure TForm1.FormClick(Sender: TObject);
begin
    Screen.ActiveForm.Color := clYellow;
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TScreen
```

ActiveMDIChild

Descrição

A propriedade ActiveMDIChild é definida como uma variável do tipo TForm, que retorna a janela-filha que possui o foco em uma aplicação MDI. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Você não pode alterar a propriedade ActiveMDIChild do componente, mas pode usar o seu resultado. O trecho de código abaixo faz com que, quando o usuário clicar sobre um botão Button do tipo TButton, a janela-filha ativa de uma aplicação MDI assuma a cor amarela.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    YellowForm: TForm;
begin
    YellowForm := Form1.ActiveMDIChild;
     YellowForm.Color := clYellow;
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TForm
```

ActivePage

Descrição

A propriedade ActivePage é definida como uma variável do tipo string, que retorna a página exibida pelo componente.

Para componentes do tipo TPageControl, é definida como uma variável do tipo TTabSheet, que retorna a página exibida pelo componente.

Exemplo

Você pode alterar a propriedade ActivePage do componente diretamente no Object Inspector ou mediante uma linha de código, como:

```
Notebook1.ActivePage := 'Primeira página';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPageControl, TNotebook e TTabbedNotebook
```

Durante a execução do aplicativo:

```
TPageControl, TNotebook e TTabbedNotebook
```

AliasName

Descrição

A propriedade AliasName é definida como uma variável do tipo TSymbolstr que especifica seu alias no Borland Database Engine (BDE).

Exemplo

Você pode alterar a propriedade AliasName diretamente no Object Inspector ou mediante uma linha de código, como:

```
TDatabase1.AliasName := 'USER';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDatabase
```

Durante a execução do aplicativo:

```
TDatabase
```

Align

Descrição

A propriedade Align é declarada como uma variável do tipo TAlign e determina como deve ser o alinhamento do controle em relação ao seu controle-pai (aquele que o contém).

Tabela de Valores:

Valor	Significado
alNone	O componente permanece no local em que foi colocado no formulário (valor default).
alTop	O componente se desloca para o topo do componente-pai e é automaticamente redimensionado, de forma a ocupar toda a largura do componente-pai, mantendo-se a sua altura original.
alBottom	O componente se desloca para a base do componente-pai e é automaticamente redimensionado, de forma a ocupar toda a largura do componente-pai, mantendo-se a sua altura original.
alLeft	O componente se desloca para a extremidade esquerda do componente-pai e é automaticamente redimensionado, de forma a ocupar toda a altura do componente-pai, mantendo-se a sua largura original.
alRight	O componente se desloca para a extremidade esquerda do componente-pai e é automaticamente redimensionado, de forma a ocupar toda a altura do componente-pai, mantendo-se a sua largura original.
alClient	O componente é redimensionado para ocupar toda a área-cliente do componente-pai.

Se o componente-pai é redimensionado, as dimensões do componente considerado serão automaticamente ajustadas, em correspondência ao valor especificado na propriedade Align.

Exemplo

Coloque um componente Panel em um formulário vazio e defina a sua propriedade Align como alTop. Em seguida, coloque um componente Tabbed Notebook *dentro* do componente Panel e defina sua propriedade Align como alClient.

É importante notar que, nesse caso, o componente-pai do componente Panel é o formulário Form1, enquanto que o componente-pai de Tabbed Notebook é o componente Panel. Conseqüentemente, a área-cliente a ser ocupada pelo componente Tabbed Notebook é a do componente Panel e não a do formulário Form1.

Componentes aos quais se aplica:

Na fase de projeto:

```
TBevel, TDBGrid, TDBRadioGroup, TDecisionPivot, TDirectoryListBox, TDrawGrid, TFileListBox,
TGroupBox, THeader, THeaderControl, TImage, TIWDBGrid, TLabel, TListBox, TListView, TMaskEdit,
TMemo, TNotebook, TOutline, TPaintBox, TPageControl, TPanel, TProgressBar, TRadioGroup, TRichEdit,
TScrollBox, TStatusBar, TStringGrid, TTabControl, TTabbedNotebook, TTabSet e TTreeView
```

Durante a execução do aplicativo:

```
TBevel, TDBGrid, TDBRadioGroup, TDecisionPivot, TDirectoryListBox, TDrawGrid, TFileListBox,
TGroupBox, THeader, THeaderControl, TImage, TIWDBGrid, TLabel, TListBox, TListView, TMaskEdit,
TMemo, TNotebook, TOutline, TPaintBox, TPageControl, TPanel, TProgressBar, TRadioGroup, TRichEdit,
TScrollBox, TStatusBar, TStringGrid, TTabControl, TTabbedNotebook, TTabSet e TTreeView
```

AlignButton

Descrição

Essa propriedade é declarada como uma variável do tipo TUDAlignButton e determina a posição do componente em relação ao controle definido na sua propriedade Associate.

Tabela de Valores:

Valor	Significado
udLeft	O componente permanece do lado esquerdo do controle associado.
udRight	O componente permanece do lado direito do controle associado.

Componentes aos quais se aplica:

Na fase de projeto:

TUpDown

Durante a execução do aplicativo:

TUpDown

Alignment

Descrição

Para controles dos tipos TCheckBox, TDBCheckBox e TRadioButton, a propriedade Alignment é uma variável do tipo TLeftRight que determina como deve ser o alinhamento do texto da propriedade Caption do componente.

Para componentes dos tipos TPopupMenu, essa propriedade é definida como uma variável do tipo TPopupAlignment, que determina onde o menu flutuante deve aparecer quando o usuário selecionar o botão direito do mouse.

Para componentes do tipo TBCDField, TBooleanField, TCurrencyField, TDateField, TDateTimeField, TDRLabel, TFloatField, TIntegerField, TSmallintField, TStringField, TTimeField, TWordField, THeaderSection, TParaAttributes, TStatusPanel, TListColumn, Tcolumn, TColumnTitle, TLabel, TMemo, TPanel, TDecisionPivot e TRichEdit, a propriedade Alignment é declarada como uma variável do tipo TAlignment e define como deve ser o alinhamento do texto no componente. Para esses componentes, essa propriedade só está disponível durante a execução do aplicativo.

Tabela de Valores para Controles dos tipos TCheckBox, TDBCheckBox e TRadioButton:

Valor	Significado
taLeftJustify	O texto é exibido do lado esquerdo do controle.
taRightJustify	O texto é exibido do lado direito do controle.

Tabela de Valores para Componentes dos tipos TBCDField, TBooleanField, TCurrencyField, TDateField, TDateTimeField, TFloatField, TIntegerField, TSmallintField, TStringField, TTimeField e TWordField, THeaderSection, TParaAttributes, TStatusPanel, TListColumn, Tcolumn, TColumnTitle, TLabel, TMemo, TPanel, TDecisionPivot e TRichEdit:

Valor	Significado
taCenter	O texto é exibido centralizado no controle.
taLeftJustify	O texto é exibido do lado esquerdo do controle.
taRightJustify	O texto é exibido do lado direito do controle.

Tabela de Valores para Componentes do tipo TPopupMenu:

Valor	Significado
paLeftta	O menu flutuante é exibido com o seu canto superior esquerdo sob o ponteiro do mouse.
paCenter	O menu flutuante é exibido com o centro da sua borda superior sob o ponteiro do mouse.
paRight	O menu flutuante é exibido com o seu canto superior direito sob o ponteiro do mouse.

Tabela de Valores para Componentes do tipo TQRLabel:

Valor	Significado
taLeft	O texto é exibido do lado esquerdo do controle.
taCenter	O texto é exibido centralizado no controle.
taRight	O texto é exibido do lado direito do controle.

Exemplo

Você pode alterar a propriedade Alignment do componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
CheckBox1.Alignment:= taLeftJustify;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRLabel, TPopupMenu, TLabel, TMemo, TPanel, TRichEdit, TCheckBox, TDBCheckBox e TRadioButton
```

Durante a execução do aplicativo:

```
TQRLabel, TPopupMenu, TCheckBox, TRadioButton, TBCDField, TBooleanField, TCurrencyField, TDateField,
TDateTimeField, TDBCheckBox, TFloatField, TIntegerField, TSmallintField, TStringField, TTimeField,
TWordField, THeaderSection, TParaAttributes, TStatusPanel, TListColumn, TColumn, TColumnTitle,
TLabel, TMemo, TPanel e TRichEdit
```

ALIGNTOBAND

Descrição

A propriedade AlignToBand é declarada como uma variável booleana, que especifica se o valor definido na propriedade Alignment será ou não aplicado ao componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRLabel
```

Durante a execução do aplicativo:

```
TQRLabel
```

Allocation

Descrição

Esta propriedade é declarada como uma variável do tipo inteiro longo, e define o número de páginas configuradas para o banco de dados.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDatabaseInfo
```

Durante a execução do aplicativo:

```
TIBDatabaseInfo
```

AllocBy

Descrição

Para controles dos tipos TListView e TImageList, a propriedade AllocBy é uma variável inteira que prepara o controle para a adição de um grande número de itens.

Exemplo

Você pode alterar a propriedade AllocBy diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
ListView1.AllocBy:= 10;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TListView e TImageList
```

Durante a execução do aplicativo:

```
TListView e TImageList
```

AllowAllUp

Descrição

A propriedade AllowAllUp é uma variável do tipo booleano que determina se, em um grupo de componentes do tipo TSpeedButton, pode ocorrer a situação em que nenhum deles esteja selecionado (o grupo de um componente TSpeedButton é definido pela sua propriedade GroupIndex). Alterar essa propriedade para um dos componentes do grupo altera também seu valor para os demais componentes.

Exemplo

Você pode alterar a propriedade AllowAllUp diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
SpeedButton1.AllowAllUp := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TSpeedButton
```

Durante a execução do aplicativo:

```
TSpeedButton
```

ALLOWDELETE

Descrição

A propriedade AllowDelete é uma variável do tipo booleana que determina se o usuário pode deletar o registro corrente usando a combinação de teclas Ctrl + Delete.

Exemplo

Você pode alterar a propriedade AllowDelete diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
DBCtrlGrid1.AllowDelete:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBCtrlGrid
```

Durante a execução do aplicativo:

```
TDBCtrlGrid
```

ALLOWGRAYED

Descrição

A propriedade AllowGrayed é uma variável do tipo booleana que determina se o controle possuirá dois ou três estados. Se for igual a True, o controle poderá possuir três estados: cbChecked, cbUnchecked e cbGrayed. A diferença entre os estados cbGrayed e cbChecked é que no primeiro caso a marca de verificação aparece com uma cor cinza.

Exemplo

Você pode alterar a propriedade AllowGrayed do componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
CheckBox1.AllowGrayed := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TCheckBox e TDBCheckBox
```

Durante a execução do aplicativo:

```
TCheckBox e TDBCheckBox
```

ALLOWINPLACE

Descrição

A propriedade AllowInPlace é uma variável do tipo booleano que determina se um objeto OLE pode ser ativado.

Exemplo

Você pode alterar a propriedade AllowInPlace diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
OLEContainer1.AllowInPlace := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOLEContainer
```

Durante a execução do aplicativo:

```
TOLEContainer
```

ALLOWINSERT

Descrição

A propriedade AllowInsert é uma variável do tipo booleana que determina se o usuário pode inserir ou adicionar um registro usando a tecla Insert ou Ctrl+Insert, respectivamente, e desde que o registro corrente não esteja sendo editado.

Exemplo

Você pode alterar a propriedade AllowInsert diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
DBCtrlGrid1.AllowInsert:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBCtrlGrid
```

Durante a execução do aplicativo:

```
TDBCtrlGrid
```

ALLOWPANNING

Descrição

A propriedade AllowPanning é uma variável do tipo TPanningMode, que define como a rolagem do gráfico pode ser feita durante a execução do aplicativo.

Essa propriedade pode receber um dos valores descritos na tabela a seguir.

Valor	Significado
pmNone	Não permite qualquer tipo de rolagem.
pmHorizontal	Permite rolagem do gráfico apenas na direção horizontal.
pmVertical	Permite rolagem do gráfico apenas na direção vertical.
pmBoth	Permite rolagem do gráfico nas duas direções.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart
```

Durante a execução do aplicativo:

```
TChart, TDBChart
```

AllowResize

Descrição

A propriedade AllowResize é uma variável booleana que define se o usuário pode ou não redimensionar as seções de um componente do tipo THeader.

Exemplo

Você pode alterar a propriedade AllowResize de um componente diretamente no Object Inspector ou através de uma linha de código como:

```
Header1.AllowResize := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
THeader
```

Durante a execução do aplicativo:

```
THeader
```

AllowSinglePoint

Descrição

A propriedade AllowSinglePoint é uma variável do tipo booleano que determina se a série representada pelo objeto deve ser exibida ainda que contenha um único ponto.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TChartSeries
```

AllowZoom

Descrição

A propriedade AllowZoom é uma variável booleana que define se o usuário pode ou não aplicar o recurso de zoom ao componente durante a execução do aplicativo.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart
```

Durante a execução do aplicativo:

```
TChart, TDBChart
```

ArrowKeys

Descrição

Essa propriedade é declarada como uma variável booleana que define se as teclas de seta para cima e seta para baixo podem ser usadas para alterar o valor armazenado na propriedade Position do componente.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
UpDown1.ArrowKeys:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TUpDown
```

Durante a execução do aplicativo:

```
TUpDown
```

AsBoolean

Descrição

A propriedade AsBoolean é uma variável booleana que realiza uma conversão de tipo.

Para componentes do tipo TBooleanField, pode ser usada para ler ou definir o valor do campo. Para componentes do tipo TStringField, AsBoolean retorna True se o texto armazenado começa com as letras "Y", "y", "T" ou "t" (de "Yes" ou "True"); se não retorna False. Para atribuir um valor ao campo, use 'T' ou 'F'. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir verifica se a string armazenada em um campo chamado Telefone tem um texto armazenado que começa com "Y", "y", "T" ou "t", exibindo uma mensagem em caso positivo. Essa propriedade só está disponível durante a execução do aplicativo.

```
if Table1.FieldByName('Telefone').AsBoolean then ShowMessage('Verdadeiro');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBooleanField e TStringField
```

AsDateTime

Descrição

A propriedade AsDateTime é declarada como uma variável do tipo TDateTime que realiza uma conversão de tipo. Para componentes dos tipos TDateField, TDateTimeField e TTimeField, pode ser usada para ler ou definir o valor do campo. Para componentes do tipo TStringField, faz a conversão de uma string em TDateTime na leitura de um campo e de TDateTime para string no sentido inverso. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir converte uma string em uma variável do tipo TDateTime antes de escrever o seu valor em um campo.

```
Table1.FieldByName(Variavel_Time).AsDateTime:= StrToDateTime(Now);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TDateField, TDateTimeField, TStringField e TTimeField
```

AsFloat

Descrição

A propriedade AsFloat é declarada como uma variável do tipo Double que realiza uma conversão de tipo. Para componentes dos tipos TBCDField, TCurrencyField e TFloatField, pode ser usada para ler ou definir o valor do campo. Para componentes do tipo TStringField, faz a conversão de uma string em

Double na leitura de um campo e de Double para string no sentido inverso. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir converte uma string em uma variável do tipo Double antes de escrever o seu valor em um campo.

```
Table1.FieldByName(Variavel_Float).AsFloat:= StrToFloat(string);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TCurrencyField, TFloatField e TStringField
```

AsInteger

Descrição

A propriedade AsInteger é declarada como uma variável do tipo inteiro longo (Longint) que realiza uma conversão de tipo. Para componentes dos tipos TIntegerField, TSmallintField e TWordField pode ser usada para ler ou definir o valor do campo. Para componentes do tipo TStringField, faz a conversão de uma string em Longint na leitura de um campo e de Longint para string no sentido inverso. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir converte uma string em uma variável do tipo Double antes de escrever o seu valor em um campo.

```
Table1.FieldByName(Variavel_Inteira).AsInteger:= StrToInt(string);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TIntegerField, TSmallintField, TStringField e TWordField
```

Associate

Descrição

A propriedade Associate é declarada como uma variável do tipo TWinControl que define o controle associado ao componente e que refletirá o valor armazenado na sua propriedade Position.

Exemplo

Você pode alterar o valor da propriedade Associate de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
UpDown1.Associate:= Edit1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TUpDown
```

Durante a execução do aplicativo:

```
TUpDown
```

AsString

Descrição

A propriedade AsString é declarada como uma variável do tipo string que realiza uma conversão de tipo.

Para componentes dos tipos TAutoIncField e TStringField, pode ser usada para ler ou definir o valor do campo.

Para componentes dos tipos TBCDField, TCurrencyField, TDateField, TDateTimeField, TFloatField, TIntegerField, TSmallintField, TTimeField e TWordField faz a conversão do tipo apropriado em uma string na leitura de um campo e de string para o tipo apropriado no sentido inverso. Essa propriedade só está disponível durante a execução do aplicativo.

Para componentes do tipo TBooleanField, define AsString como True se o texto começa com a letra "Y", "y", "T" ou "t"; se não define como False. Na leitura, retorna 'T' ou 'F'.

Para os demais tipos de componentes, essa propriedade deve ser usada apenas para leitura do campo e conversão no tipo apropriado, gerando um erro de exceção caso se tente realizar uma operação de atribuição/gravação.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField, TStringField,
TTimeField, TVarBytesField e TWordField
```

AsText

Descrição

Essa propriedade retorna o conteúdo do Clipboard como uma string. Se o conteúdo do Clipboard não for um texto, gera um erro de exceção. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir exibe em um componente Label1 do tipo Tlabel o texto armazenado no Clipboard.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TClipboard
```

AutoActivate

Descrição

A propriedade AutoActivate é uma variável do tipo TAutoActivate que determina como um objeto pode ser ativado em um componente do tipo TOLEContainer.

Tabela de Valores:

Valor	Significado
aaManual	O objeto OLE deve ser ativado manualmente, definindo-se a sua propriedade Active igual a True.
aaGetFocus	O objeto OLE se torna ativo quando o componente que o contém recebe o foco da aplicação.
aaDoubleClick	O objeto OLE se torna ativo quando o componente que o contém possui o foco da aplicação e o usuário pressiona a tecla Enter ou dá um duplo clique sobre o mesmo.

Exemplo

Você pode alterar a propriedade AutoActivate de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
OLEContainer1.AutoActivate := aaManual;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOLEContainer
```

Durante a execução do aplicativo:

```
TOLEContainer
```

AUTOCALCFIELDS

Descrição

A propriedade AutoCalcFields é uma variável booleana que determina se o evento OnCalcFields deve ser disparado quando a aplicação carrega um novo registro do banco de dados associado.

Exemplo

Você pode alterar a propriedade AutoCalcFields de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Query1.AutoCalcFields := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TDecisionQuery, TIBDataset, TIBQuery,
TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery, TDecisionQuery e TStoredProc
```

Durante a execução do aplicativo:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TDecisionQuery, TIBDataset, TIBQuery,
TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery, TDecisionQuery e TStoredProc
```

AUTODISPLAY

Descrição

A propriedade AutoDisplay é uma variável booleana que determina se o conteúdo do controle deve ser exibido automaticamente.

Exemplo

Você pode alterar a propriedade AutoDisplay de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
DBImage1.AutoDisplay := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBImage e TDBMemo
```

Durante a execução do aplicativo:

```
TDBImage e TDBMemo
```

AutoEdit

Descrição

Essa propriedade é declarada como uma variável booleana que define se os controles conectados a um componente do tipo TDataSource estarão ou não em modo de edição.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
DataSource1.AutoEdit:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDataSource
```

Durante a execução do aplicativo:

```
TDataSource
```

AutoEnable

Descrição

A propriedade AutoEnable é uma variável booleana que determina se o componente pode habilitar e desabilitar automaticamente seus botões.

Exemplo

Você pode alterar a propriedade AutoEnable de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
MediaPlayer1.AutoEnable := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMediaPlayer
```

Durante a execução do aplicativo:

```
TMediaPlayer
```

AutoMerge

Descrição

A propriedade AutoMerge é uma variável booleana que determina se os menus de diversos formulários devem ser combinados automaticamente.

Exemplo

Você pode alterar a propriedade AutoMerge de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
MainMenu1.AutoMerge := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMainMenu
```

Durante a execução do aplicativo:

```
TMainMenu
```

AutoOpen

Descrição

A propriedade AutoOpen é uma variável booleana que determina se o componente pode tentar abrir automaticamente o dispositivo multimídia associado quando a aplicação é executada.

Exemplo

Você pode alterar a propriedade AutoOpen de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
MediaPlayer1.AutoOpen := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMediaPlayer
```

Durante a execução do aplicativo:

```
TMediaPlayer
```

AutoPopup

Descrição

A propriedade AutoPopup é uma variável booleana que determina se o menu deve aparecer quando o usuário clica com o botão direito do mouse sobre o controle que tem esse componente especificado na sua propriedade PopupMenu.

Exemplo

Você pode alterar a propriedade AutoPopup de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Popupmenu1.AutoPopup := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPopupMenu
```

Durante a execução do aplicativo:

```
TPopupMenu
```

AutoRewind

Descrição

A propriedade AutoRewind é uma variável booleana que determina se o componente deve ser rebobinado automaticamente durante uma execução ou gravação.

Exemplo

Você pode alterar a propriedade AutoRewind de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
MediaPlayer1.AutoRewind := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMediaPlayer
```

Durante a execução do aplicativo:

```
TMediaPlayer
```

AutoScroll

Descrição

Para componentes dos tipos TForm, TFrame, e TScrollBox, a propriedade AutoScroll é uma variável booleana que determina se o controle exibirá automaticamente as barras de rolagem quando for grande o suficiente para exibir todos os controles que contém.

Para componentes do tipo TTabSet, a propriedade AutoScroll é uma variável booleana que determina se o controle exibirá automaticamente botões de rolagem quando for grande o suficiente para exibir todos os controles que contém.

Exemplo

Você pode alterar a propriedade AutoScroll de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Form1.AutoScroll := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm, TFrame, TScrollBox e TTabSet
```

Durante a execução do aplicativo:

```
TForm, TFrame, TScrollBox e TTabSet
```

AutoSelect

Descrição

A propriedade AutoSelect é uma variável booleana que determina se o texto exibido pelo controle será selecionado quando este receber o foco da aplicação.

Exemplo

Você pode alterar a propriedade AutoSelect de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Edit1.AutoSelect := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBEdit, TDBLookupCombo, TEdit e TMaskEdit
```

Durante a execução do aplicativo:

```
TDBEdit, TDBLookupCombo, TEdit e TMaskEdit
```

AutoSize

Descrição

Para componentes dos tipos TDBText e TLabel, a propriedade AutoSize é uma variável booleana que determina se o controle será automaticamente redimensionado para acomodar o texto definido na sua propriedade Caption.

Para componentes dos tipos TDBEdit, TDBLookupCombo, TEdit, THotKey e TMaskEdit, a propriedade AutoSize é uma variável booleana que determina se a altura do controle será redimensionada quando o tamanho da fonte de texto for alterado.

Para componentes dos tipos TDBImage e TImage, a propriedade AutoSize é uma variável booleana que determina se o controle será automaticamente redimensionado para acomodar a imagem definida na sua propriedade Picture.

Exemplo

Você pode alterar a propriedade AutoSize de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Edit1.AutoSize := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBText, TLabel, TDBEdit, TDBLookupCombo, TEdit, THotKey, TMaskEdit, TDBImage e TImage
```

Durante a execução do aplicativo:

```
TDBText, TLabel, TDBEdit, TDBLookupCombo, TEdit, THotKey, TMaskEdit, TDBImage e TImage
```

AxisVisible

Descrição

A propriedade AxisVisible é uma variável booleana que define se todos os quatro eixos do gráfico (dois horizontais e dois verticais) devem ou não ser exibidos. Você pode definir essa propriedade como False e, posteriormente, habilitar individualmente a exibição de cada eixo.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart
```

Durante a execução do aplicativo:

```
TChart, TDBChart
```

BackColor

Descrição

A propriedade BackColor é declarada como uma variável do tipo TColor e define a cor de fundo do gráfico.

Para o componente THTML, define a cor de fundo da página HTML exibida pelo componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart, THTML
```

Durante a execução do aplicativo:

```
TChart, TDBChart, THTML
```

BackgroundColor

Descrição

A propriedade BackgroundColor é declarada como uma variável do tipo TColor e define a cor de fundo do componente.

Exemplo

Você pode alterar a propriedade BackgroundColor de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
TabSet1.BackgroundColor := clRed;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTabSet
```

Durante a execução do aplicativo:

```
TTabSet
```

BackImage

Descrição

A propriedade BackImage é declarada como um objeto da classe TPicture, e define a imagem de fundo a ser exibida juntamente com o gráfico.

Para o componente THTML, define uma URL que armazena a imagem de fundo a ser exibida em uma página Web.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart, THTML
```

Durante a execução do aplicativo:

```
TChart, TDBChart, THTML
```

BevelInner

Descrição

A propriedade BevelInner é declarada como uma variável do tipo TPanelBevel e define o estilo do chanfro interno de um componente do tipo TPanel.

Tabela de Valores:

Valor	Significado
bvNone	Não existe chanfro interno.
bvLowered	O chanfro interno existe na forma de um entalhe (depressão).
bvRaised	O chanfro interno é saliente (elevado).

Exemplo

Você pode alterar a propriedade BevelInner de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Panel1.BevelInner := bvNone;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart, TDecisionGraph, TPanel, TDecisionGrid, TDecisionPivot
```

Durante a execução do aplicativo:

```
TChart, TDBChart, TDecisionGraph, TPanel, TDecisionGrid, TDecisionPivot
```

BEVELOUTER

Descrição

A propriedade BevelOuter é declarada como uma variável do tipo TPanelBevel e define o estilo do chanfro externo de um componente do tipo TPanel.

Tabela de Valores:

Valor	Significado
bvNone	Não existe chanfro externo.
bvLowered	O chanfro externo existe na forma de um entalhe (depressão).
bvRaised	O chanfro externo é saliente (elevado).

Exemplo

Você pode alterar a propriedade BevelOuter de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Panel1.BevelOuter := bvNone;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart, TDecisionGraph, TPanel, TDecisionGrid, TDecisionPivot
```

Durante a execução do aplicativo:

```
TChart, TDBChart, TDecisionGraph, TPanel, TDecisionGrid, TDecisionPivot
```

BEVELWIDTH

Descrição

A propriedade BevelWidth é declarada como uma variável do tipo TBevelWidth e define a espessura, em pixels, da linha que separa os chanfros interno e externo em um componente do tipo TPanel.

Exemplo

Você pode alterar a propriedade BevelWidth de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Panel1.BevelWidth := 4;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart, TDecisionGraph, TPanel, TDecisionGrid, TDecisionPivot
```

Durante a execução do aplicativo:

```
TChart, TDBChart, TDecisionGraph, TPanel, TDecisionGrid, TDecisionPivot
```

Bitmap

Descrição

A propriedade Bitmap é declarada como uma variável do tipo TBitmap e define um padrão de preenchimento para um objeto do tipo TBrush.

Para componentes do tipo TPicture, essa propriedade é declarada como uma variável do tipo TBitmap e representa o conteúdo do objeto na forma de um Bitmap.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBitmap e TPicture
```

BkColor

Descrição

Essa propriedade é declarada como uma variável do tipo TColor e especifica a cor de fundo das imagens a serem exibidas.

Exemplo

Você pode alterar a propriedade BkColor diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
ImageList1.BkColor:= clRed;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TImageList
```

Durante a execução do aplicativo:

```
TImageList
```

BlendColor

Descrição

Essa propriedade é declarada como uma variável do tipo TColor e especifica a cor de primeiro plano das imagens a serem exibidas.

Exemplo

Você pode alterar a propriedade BlendColor diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
ImageList1.BlendColor:= clYellow;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TImageList
```

Durante a execução do aplicativo:

```
TImageList
```

BlobSize

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro e especifica o número de bytes armazenados no registro corrente para um campo do tipo blob.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBlobField
```

BOF

Descrição

A propriedade BOF é uma variável booleana que determina se um banco de dados está no seu primeiro registro. Isto ocorre após uma tabela ser aberta e ocorrer uma chamada aos eventos First ou Prior. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataset, TDecisionQuery, TIBDataset, TIBQuery,
TIBStoredProc, TIBSQL, TIBTable, TIBTransaction, TTable, TQuery, TDecisionQuery e TStoredProc
```

BORDERICONS

Descrição

A propriedade BorderIcons é declarada como uma variável do tipo TBorderIcons e consiste em um conjunto de variáveis que determina os ícones a serem exibidos na barra de título de um formulário.

Tabela de Valores:

Valor	Significado
biSystemMenu	O formulário possui um menu de sistema.
biMinimize	O formulário possui um botão de minimização.
biMaximize	O formulário possui um botão de maximização.

Exemplo

Você pode alterar a propriedade BorderIcons de um formulário diretamente no Object Inspector ou através de uma linha de código, como:

```
Form1.BorderIcons := [biSystemMenu,biMinimize,biMaximize] ;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TForm
```

BORDERSTYLE

Descrição

Para componentes do tipo TForm, essa propriedade é declarada como uma variável do tipo TFormBorderStyle, que determina o estilo de borda do componente.

Para os demais componentes, a propriedade BorderStyle é declarada como uma variável do tipo TBorderStyle, que determina se o componente terá ou não uma borda.

Tabela de Valores para componentes do Tipo TForm:

Valor	Significado
bsDialog	Borda não redimensionável, comum em quadros de diálogo.
bsSingle	Borda simples e redimensionável.
bsNone	Borda invisível, não redimensionável, sem botões de maximização, minimização e menu de sistema.
bsSizeable	Borda padrão redimensionável.

Para janelas cuja propriedade FormStyle é fsMDIChild, bsDialog ou bsNone e não afeta sua aparência.

Tabela de Valores para os demais componentes:

Valor	Significado
bsNone	Sem borda
bsSingle	Borda Simples

Exemplo

Você pode alterar a propriedade AutoSize de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Edit1.BorderStyle:= bsSingle;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDrawGrid, TEdit,
TForm, THeader, TIWDBGrid, TListBox, TListView, TMaskEdit, TMemo, TOutline, TPanel, TDecisionPivot,
TRichEdit, TScrollBox, TStringGrid e TTreeView
```

Durante a execução do aplicativo:

```
TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDrawGrid, TEdit,
TForm, THeader, TIWDBGrid, TListBox, TListView, TMaskEdit, TMemo, TOutline, TPanel, TDecisionPivot,
TRichEdit, TScrollBox, TStringGrid e TTreeView
```

BorderWidth

Descrição

A propriedade BorderWidth é declarada como uma variável do tipo TBorderWidth e define a espessura, em pixels, da borda de um componente dos tipos TPanel e TDecisionPivot.

Exemplo

Você pode alterar a propriedade BorderWidth de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Panel1.BorderWidth := 4;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPanel, TDecisionPivot.
```

Durante a execução do aplicativo:

```
TPanel, TDecisionPivot
```

BottomAxis

Descrição

A propriedade BottomAxis é declarada como um objeto da classe TChartAxis, e representa o eixo horizontal inferior do gráfico exibido no componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart
```

Durante a execução do aplicativo:

```
TChart, TDBChart
```

BoundsRect

Descrição

A propriedade BoundsRect é uma variável do tipo TRect, cujos campos estabelecem, nas coordenadas do sistema definido pelo controle-pai, o retângulo circunscrito ao controle atual.

Os campos da propriedade BoundsRect (como qualquer variável do tipo TRect) são: Left, Top, Right e Bottom.

Tabela de Valores:

Valor	Significado
BoundsRect.Left	Valor da coordenada **X** da extremidade superior esquerda do retângulo circunscrito ao controle.
BoundsRect.Top	Valor da coordenada **Y** da extremidade superior esquerda do retângulo circunscrito ao controle.
BoundsRect.Right	Valor da coordenada **X** da extremidade inferior direita do retângulo circunscrito ao controle.
BoundsRect.Bottom	Valor da coordenada **Y** da extremidade inferior direita do retângulo circunscrito ao controle.

Exemplo

Com relação às propriedades Left, Top, Right e Bottom de um formulário, podemos afirmar que:

```
BoundsRect.Left = Left
BoundsRect.Top = Top
BoundsRect.Right = Left + Width
BoundsRect.Bottom = Top + Height
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os controles.

Break

Descrição

Essa propriedade é declarada como uma variável do tipo TMenuBreak e permite dividir um menu em colunas.

Tabela de Valores:

Valor	Significado
mbNone	O menu não é dividido em colunas.
mbBarBreak	A quebra ocorre com o item de menu aparecendo no topo da nova coluna. Uma barra vertical separa as colunas.
mbBreak	A quebra ocorre com o item de menu aparecendo no topo da nova coluna. Nesse caso não surge uma barra vertical separando as colunas.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
MenuItem1.Break := mbNone;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMenuItem
```

Durante a execução do aplicativo:

```
TMenuItem
```

Brush

Descrição

Essa propriedade é declarada como uma variável do tipo TBrush que especifica a cor e o padrão de preenchimento usados para pintar formas gráficas, e como cor e pano de fundo.

No caso de controles, especifica a cor e o padrão de preenchimento usados como cor e pano de fundo.

Exemplo

Você pode criar um formulário Form1 do tipo TForm com a cor amarela e hachuras em diagonal com o seguinte trecho de código:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Form1.Brush.Color := clYellow;
  Form1.Brush.Style := bsFDiagonal;
end;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TShape
```

Durante a execução do aplicativo:

Todos os controles e Tcanvas.

ButtonAutoSize

Descrição

Essa propriedade é declarada como uma variável booleana, e define se os botões exibidos no componente devem ser redimensionados automaticamente.

Componentes aos quais se aplica:

Na fase de projeto:

`TDecisionPivot`

Durante a execução do aplicativo:

`TDecisionPivot`

ButtonHeight

Descrição

Essa propriedade é declarada como uma variável inteira e define o valor, em pixels, da altura dos botões exibidos no componente.

Componentes aos quais se aplica:

Na fase de projeto:

`TDecisionPivot`

Durante a execução do aplicativo:

`TDecisionPivot`

ButtonSpacing

Descrição

Essa propriedade é declarada como uma variável inteira e define o valor, em pixels, do espaçamento dos botões exibidos no componente.

Componentes aos quais se aplica:

Na fase de projeto:

`TDecisionPivot`

Durante a execução do aplicativo:

`TDecisionPivot`

ButtonWidth

Descrição

Essa propriedade é declarada como uma variável inteira e define o valor, em pixels, da largura dos botões exibidos no componente.

Componentes aos quais se aplica:

Na fase de projeto:

`TDecisionPivot`

Durante a execução do aplicativo:

`TDecisionPivot`

BytesRcvd

Descrição

Essa propriedade é declarada como uma variável inteira e define o número de bytes já recebidos em uma conexão.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TNMFTP, TNMHTTP, TNMNNTP, TNMPOP3, TPowerSock
```

BytesSent

Descrição

Essa propriedade é declarada como uma variável inteira e define o número de bytes já enviados em uma conexão.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TNMFTP, TNMHTTP, TNMNNTP, TPowerSock
```

BytesTotal

Descrição

Essa propriedade é declarada como uma variável inteira e define o número total de bytes a serem recebidos em uma conexão.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TNMFTP, TNMHTTP, TNMNNTP, TNMPOP3, TPowerSock
```

CachedSize

Descrição

Essa propriedade é definida como uma variável inteira, e define o número de registros armazenados na memória local pelo componente. Indica portanto o número de registros carregados quando a conexão através do componente se torna ativa.

Componentes aos quais se aplica:

Na fase de projeto:

```
TClientDataSet, TADODataset, TADOTable, TADOQuery, TIBTable
```

Durante a execução do aplicativo:

```
TClientDataSet, TADODataset, TADOTable, TADOQuery, TIBTable
```

CachedUpdates

Descrição

Esta propriedade é declarada como uma variável do tipo booleana e indica se o recurso de Cached Updates está sendo utilizado pelo componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TClientDataSet, TADODataset, TADOTable,  TIBTable
```

Durante a execução do aplicativo:

```
TClientDataSet, TADODataset, TADOTable,  TIBTable
```

Calculated

Descrição

A propriedade Calculated é declarada como uma variável do tipo booleana e indica se o valor do campo foi calculado em um evento OnCalcFields. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir exibe uma mensagem informando se o campo foi calculado num evento OnCalcFields, onde WordField é uma variável do tipo TWordField.

```
if WordField.Calculated:= True then ShowMessage('Valor calculado em evento');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField, TStringField,
TTimeField, TVarBytesField e TWordField
```

Cancel

Descrição

A propriedade Cancel é declarada como uma variável do tipo booleano e indica se um controle do tipo TButton ou TBitBtn associa o seu evento OnClick ao pressionamento da tecla Esc.

Exemplo

Coloque um botão chamado Button1 num formulário chamado Form1 e defina a sua propriedade Cancel como True no Object Inspector. No evento OnClick de Button1 inclua a seguinte linha de código:

```
if Form1.Color := clYellow then Form1.Color := clRed else Form1.Color := clYellow;
```

Execute o aplicativo e pressione seguidamente a tecla Esc e você verá a cor do formulário alternar entre vermelho e amarelo.

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn e TButton
```

Durante a execução do aplicativo:

```
TBitBtn e TButton
```

Canceled

Descrição

Essa propriedade é declarada como uma variável booleana e define se o método Cancel do componente foi acionado.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TQRPrinter
```

CanModify

Descrição

A propriedade CanModify é declarada como uma variável do tipo booleana, e indica se o valor do campo pode ser alterado. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário. Para componentes do tipo TTable, TQuery, TDecisionQuery e TStoredProc, define se um campo do banco de dados pode ser alterado.

Exemplo

O trecho de código a seguir exibe uma mensagem informando se o valor do campo pode ser modificado, onde WordField é uma variável do tipo TWordField.

```
if WordField.CanModify := True then ShowMessage('Valor pode ser modificado');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField, TStringField,
TTimeField, TVarBytesField, TWordField, TClientDataSet, TADODataset, TADOTable, TADOQuery, TTable,
TQuery, TDecisionQuery e TStoredProc
```

Canvas

Descrição

A propriedade Canvas é declarada como uma variável do tipo TCanvas e fornece uma área gráfica na qual o programa pode realizar desenhos durante a sua execução. Essa propriedade só está disponível durante a execução do aplicativo.

Para componentes dos tipos TForm, TImage e TPaintBox, você desenha com a propriedade Canvas quando é disparado um evento OnPaint.

Para componentes dos tipos TComboBox, TDirectoryListBox, TFileListBox, TListBox e TOutline você desenha com a propriedade Canvas quando é disparado um evento OnDrawItem.

Para componentes dos tipos TDBGrid, TDBCtrlGrid, TDrawGrid, TIWDBGrid e TStringGrid, você desenha com a propriedade Canvas quando é disparado um evento OnDrawCell ou OnDrawDataCell.

Para componentes dos tipos TPrinter e TQRPrinter, você desenha com a propriedade Canvas através dos métodos Draw, StretchDraw e CopyRect (desde que a impressora associada ao objeto TPrinter seja capaz de imprimir gráficos).

Para componentes dos tipos TChart e TDBChart, você desenha com a propriedade Canvas nos procedimentos associados aos eventos AfterDrawValues e BeforeDrawValues.

Para componentes dos tipos TBaseReport, TCanvasReport, TRvNDRWriter, TRVRenderPreview e TRvRenderPrinter, define a superfície de desenho do relatório.

A propriedade Canvas é por si só um objeto da GDI do Windows e tem subpropriedades como Pen, Brush, Font e Color, que também são objetos da GDI.

Exemplo

Se você quiser que o seu programa desenhe sempre um retângulo vermelho dentro de um componente PaintBox1 do tipo TPaintBox, defina o evento OnPaint de PaintBox1 da seguinte forma:

```
procedure TForm1.PaintBox1Paint(Sender: Tobject); begin
    PaintBox1.Canvas.Brush.Color:= clRed;
    PaintBox1.Canvas.Rectangle(10,10,150,150);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBaseReport, TBitmap, TCanvasReport, TComboBox, TChart, TDBChart, TDBComboBox, TDBGrid, TDBCtrlGrid,
TDBListBox, TDirectoryListBox, TDrawGrid, TFileListBox, TForm, TImage, TIWDBGrid, TListBox,
TOutline, TPaintBox, TPrinter, TQRPrinter, TRvNDRWriter, TRVRenderPreview, TRvRenderPrinter, e
TStringGrid
```

CAPABILITIES

Descrição

A propriedade Capabilities é declarada como uma variável do tipo TMPDevCapsSet e consiste num conjunto de propriedades que definem as capacidades do dispositivo multimídia corrente. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Tabela de Valores:

Valor	Significado
mpCanEject	Pode realizar operações de ejeção.
mpCanPlay	Pode executar o dispositivo.
mpCanRecord	Pode realizar operações de gravação.
mpCanStep	Pode avançar e retroceder no dispositivo.
mpUsesWindows	Usa uma janela para exibição de resultados.

Exemplo

O trecho de código a seguir verifica se o dispositivo mostra uma janela de exibição dos resultados e, em caso positivo, emite uma mensagem.

```
if mpUsesWindows in MediaPlayer1.Capabilities then ShowMessage('Dispositivo exibe janela');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

CAPACITY

Descrição

Para objetos da classe TList, a propriedade Capacity é declarada como uma variável inteira que define o espaço alocado para armazenar a lista de objetos.

Para componentes DecisionCube, a propriedade Capacity é declarada como uma variável inteira que define a memória máxima a ser alocada para o cache do componente.

Essa variável só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDecisionCube, TList
```

Caption

Descrição

A propriedade Caption é declarada como uma variável do tipo string, e indica o rótulo exibido para o componente. Para transformar um dos caracteres do rótulo em uma tecla aceleradora, basta colocar um "E comercial" - (&) antes do caractere, e o caractere será exibido sublinhado.

Para formulários, essa propriedade indica o texto exibido na sua barra de títulos, e o rótulo associado ao seu ícone, quando o controle estiver minimizado.

Para objetos da classe TAction, que representam ações definidas em um componente TactionList, essa propriedade define o texto a ser exibido nos controles e itens de menu associados.

Observação: Essa capacidade de transformar um caractere em tecla aceleradora não se aplica à propriedade Caption de um formulário.

Exemplo

Coloque um botão chamado Button1 num formulário chamado Form1 e defina a sua propriedade Caption como &Botão (o rótulo exibido será Botão) no Object Inspector. A seqüência de teclas Alt+B gera um evento OnClick de Button1.

Componentes aos quais se aplica:

Na fase de projeto:

```
TAction, TBitBtn, TButton, TCheckBox, TDBCheckBox, TDBRadioGroup, TForm, TGroupBox, TIWButton,
TIWCheckBox, TIWControl, TIWDBCheckBox, TIWDBFile, TIWFile, TIWLabel, TLabel, TPanel, TRadioButton,
TSpeedButton, TTabSheet e TMenuItem
```

Durante a execução do aplicativo:

```
TAction, TBitBtn, TButton, TCheckBox, TDBCheckBox, TDBRadioGroup, TForm, TGroupBox, TIWButton,
TIWCheckBox, TIWDBCheckBox, TIWLabel, TLabel, TPanel, TRadioButton, TSpeedButton, TTabSheet e TMenuItem
```

CaptionColor

Descrição

A propriedade CaptionColor é declarada como uma variável do tipo TColor e define a cor de fundo do texto que exibe o nome das linhas e colunas correspondentes às dimensões de dados exibidas pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDecisionGraph
```

CaptionFont

Descrição

A propriedade CaptionColor é declarada como uma variável do tipo TFont e define a fonte do texto que exibe o nome das linhas e colunas correspondentes às dimensões de dados exibidas pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDecisionGraph
```

Category

Descrição

A propriedade Category é declarada como uma variável do tipo string, e define a categoria à qual o objeto pertence. Você pode digitar seu valor diretamente no Object Inspector, quando o objeto estiver selecionado no editor de ações do Delphi.

Componentes aos quais se aplica:

Na fase de projeto:

```
TAction
```

Durante a execução do aplicativo:

```
TAction
```

Cells

Descrição

A propriedade Cells é declarada como uma array de strings onde cada item da array corresponde ao texto exibido em uma célula da grade, definida pelos índices ARow e ACol, que variam de zero até RowCount-1 e ColCount-1, respectivamente. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar o texto exibido na primeira célula da grade através de uma linha de código, como:

```
StringGrid1.Cells[0,0] := 'Primeira célula' ;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TStringGrid, TDecisionGrid
```

Center

Descrição

A propriedade Center é uma variável booleana que determina se a imagem deve ser exibida centralizada no controle. Se for igual a False, o alinhamento da imagem e do controle é feito coincidindo-se a extremidade superior esquerda.

Exemplo

Você pode alterar a propriedade Center de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Image1.Center := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TAnimate, TImage e TDBImage
```

Durante a execução do aplicativo:

```
TAnimate, TImage e TDBImage
```

ChangedCount

Descrição

A propriedade ChangedCount é declarada como uma variável inteira que armazena o número de registros adicionados à tabela definida na propriedade ChangedTableName do componente. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBatchMove
```

ChangedTableName

Descrição

A propriedade ChangedTableName é declarada como uma variável do tipo TFileName que armazena os registros que sofreram alteração devido a uma operação realizada por um componente do tipo TBatchMove.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou através de uma linha de código, como:

```
BatchMove1.ChangedTableName := 'registro.db';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBatchMove
```

Durante a execução do aplicativo:

```
TBatchMove
```

CharCase

Descrição

A propriedade CharCase é declarada como uma variável do tipo TEditCharCase que define se o texto exibido pelo componente deve aparecer apenas em letras maiúsculas, apenas em letras minúsculas ou em letras maiúsculas e minúsculas.

Tabela de Valores:

Valor	Significado
ecLowerCase	Texto exibido em letras minúsculas.
ecNormal	Texto exibido em letras maiúsculas e minúsculas.
ecUpperCase	Texto exibido em letras maiúsculas.

Exemplo

Você pode alterar a propriedade CharCase de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Edit1.CharCase := ecNormal;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBEdit, TEdit e TMaskEdit
```

Durante a execução do aplicativo:

```
TDBEdit, TEdit e TMaskEdit
```

CHECKED

Descrição

A propriedade Checked é declarada como uma variável do tipo booleana que determina se o controle está selecionado.

Tabela de Valores para controles dos tipos TCheckBox e TDBCheckBox:

Valor	Significado
True	O componente está selecionado e apresenta uma marca de verificação na cor preta.
False	O componente não está selecionado; apresenta uma marca de verificação na cor cinza ou não apresenta uma marca de verificação.

Tabela de Valores para controles do tipo TRadioButton:

Valor	Significado
True	O componente está selecionado e apresenta um círculo na cor preta.
False	O componente não está selecionado e não apresenta um círculo na cor preta.

Tabela de Valores para controles do tipo TMenuItem:

Valor	Significado
True	Uma marca de verificação aparece ao lado do item de menu, indicando que ele está selecionado.
False	Nenhuma marca de verificação aparece ao lado do item de menu, indicando que ele não está selecionado.

Para objetos da classe TAction, que representam ações definidas em um componente TactionList, essa propriedade define o texto a ser exibido nos controles e itens de menu associados.

Exemplo

Você pode alterar a propriedade Checked do componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
CheckBox1.Checked := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TAction, TCheckBox, TDBCheckBox, TIWCheckBox, TIWDBCheckBox, TMenuItem e TRadioButton
```

Durante a execução do aplicativo:

```
TAction, TCheckBox, TDBCheckBox, TIWCheckBox, TIWDBCheckBox, TMenuItem e TRadioButton
```

ClientHandle

Descrição

A propriedade ClientHandle é uma variável do tipo HWND que define um handle para a área-cliente de uma janela MDI. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que a altura da área-cliente de um formulário seja reduzida à metade cada vez que o usuário dá um clique com o mouse sobre o formulário.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     Form1.ClientHeight := Form1.ClientHeight div 2;
end;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

Todos os controles.

ClientHeight

Descrição

A propriedade ClientHeight é uma variável inteira que define a altura, em pixels, da área-cliente de um controle. Com exceção de componentes do tipo TForm, essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode usar essa propriedade para chamadas às funções da API do Windows.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os controles, TFrame e TForm.

ClientOrigin

Descrição

A propriedade ClientOrigin é declarada como uma variável do tipo TPoint que fornece, em pixels, as coordenadas do canto superior esquerdo da área-cliente de um controle, no sistema de coordenadas da tela. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que as coordenadas da origem da área-cliente de um formulário sejam exibidas em um componente Label1 do tipo TLabel.

```
Label1.Caption := 'X = '+IntToStr(Form1.ClientOrigin.X)+' Y = '+ IntToStr(Form1.ClientOrigin.Y);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os controles.

CLIENTRECT

Descrição

A propriedade ClientRect é declarada como uma variável do tipo TRect que define o tamanho, em pixels, da área-cliente de um controle. Os campos Top, Left, Bottom e Right de ClientRect definem as coordenadas do canto superior esquerdo (Left,Top) e inferior direito (Right, Bottom) da área-cliente. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir desenha uma linha do canto superior esquerdo ao canto inferior direito da área-cliente do formulário corrente.

```
with ClientRect do
begin
      Canvas.MoveTo(Left,Top);
      Canvas.LineTo(Right, Bottom);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os controles.

CLIENTWIDTH

Descrição

A propriedade ClientWidth é uma variável inteira que define a largura, em pixels, da área-cliente de um controle. Com exceção de componentes do tipo TForm, essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que a largura da área-cliente de um formulário seja reduzida à metade cada vez que o usuário dá um clique com o mouse sobre o formulário.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
          Form1.ClientWidth := Form1.ClientWidth div 2;
end;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm e TFrame
```

Durante a execução do aplicativo:

Todos os controles.

ClipRect

Descrição

A propriedade ClipRect define um retângulo de clipping na superfície do desenho. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TCanvas
```

Col

Descrição

A propriedade Col é declarada como uma variável do tipo inteiro longo (Longint) que define a que coluna pertence a célula que possui o foco. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe a coluna que possui o foco em um componente StringGrid1 do tipo TStringGrid quando o usuário seleciona uma célula com o mouse.

```
procedure TForm1.StringGrid1Click(Sender: TObject);
begin
          Label1.Caption := 'Coluna ' + IntToStr(StringGrid1.Col + 1);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDrawGrid e TStringGrid
```

ColCount

Descrição

A propriedade ColCount é declarada como uma variável do tipo inteiro longo (Longint) que define o número de colunas do controle. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe o número de colunas de um componente StringGrid1 do tipo TStringGrid quando o usuário seleciona uma célula com o mouse.

```
procedure TForm1.StringGrid1Click(Sender: Tobject);
begin
     Label1.Caption:= IntToStr(StringGrid1.ColCount);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBCtrlGrid, TDrawGrid, TDecisionGrid e TStringGrid
```

Collate

Descrição

A propriedade Collate é declarada como uma variável booleana que define se as páginas serão impressas agrupadas.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
PrintDialog1.Collate := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPrintDialog
```

Durante a execução do aplicativo:

```
TPrintDialog
```

COLOR

Descrição

A propriedade Color é declarada como uma variável do tipo TColor e define a cor de fundo de um componente.

Tabela de Valores:

Valor	Significado
clBlack	Preto
clMaroon	Marrom
clGreen	Verde
clOlive	Verde-Oliva
clNavy	Azul-Marinho
clPurple	Roxo
clTeal	Teal
clGray	Cinza
clSilver	Prata
clRed	Vermelho
clLime	Verde-Limão
clBlue	Azul
clFuchsia	Rosa
clAqua	Azul-Claro
clWhite	Branco
clBackground	Cor de fundo de janela definida no Painel de Controle do Windows.
clActiveCaption	Cor da barra de título da janela ativa, definida no Painel de Controle do Windows.
clInactiveCaption	Cor da barra de título da janela inativa, definida no Painel de Controle do Windows.
clMenu	Cor de fundo de Menu, definida no Painel de Controle do Windows.
clWindow	Cor de fundo da janela ativa, definida no Painel de Controle do Windows.
clWindowFrame	Cor do contorno das janelas, definida no Painel de Controle do Windows.

continua

continuação

Valor	Significado
clMenuText	Cor do texto em menus, definida no Painel de Controle do Windows.
clWindowText	Cor do texto em janelas, definida no Painel de Controle do Windows.
clCaptionText	Cor do texto na barra de título da janela ativa, definida no Painel de Controle do Windows.
clActiveBorder	Cor da borda da janela ativa, definida no Painel de Controle do Windows.
clInactiveBorder	Cor da borda da janela inativa, definida no Painel de Controle do Windows.
clAppWorkSpace	Cor da área de trabalho do aplicativo, definida no Painel de Controle do Windows.
clHighlight	Cor de fundo de um texto selecionado, definida no Painel de Controle do Windows.
clHighlightText	Cor de um texto selecionado, definida no Painel de Controle do Windows.
clBtnFace	Cor da face de um botão, definida no Painel de Controle do Windows.
clBtnShadow	Cor da sombra de um botão, definida no Painel de Controle do Windows.
clGrayText	Cor do texto obscuro em um botão, definida no Painel de Controle do Windows.
clBtnText	Cor do texto exibido em um botão, definida no Painel de Controle do Windows.
clInactiveCaptionText	Cor do texto da barra de título de uma janela inativa, definida no Painel de Controle do Windows.
clBtnHighlight	Cor de um botão, definida no Painel de Controle do Windows.

Exemplo

Você pode alterar a propriedade Color de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
CheckBox1.Color := ClYellow;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBrush, TFont, TPen; TCheckBox, TColorDialog, TComboBox, TDBCheckBox, TDBComboBox, TDBCtrlGrid,
TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupComboBox, TDBLookupList,
TDBLookupListBox, TDBMemo, TDBRadioGroup, TDBText, TDirectoryListBox, TDrawGrid, TDriveComboBox,
TEdit, TFileListBox, , TFrame TForm, TGroupBox, TIWCheckBox, TIWControl, TIWDBCheckBox, TIWDBEdit,
TIWDBFile, T IWDBGrid, TIWDBImage, TIWDBListBox, TIWEdit, TIWFile, TIWLabel, TIWListBox,
TIWLookupCombobox, TLabel, TListBox, TListView, TMaskEdit, TMemo, TNotebook, TOutline, TPageControl,
TPaintBox, TPanel, TQRBand, TQRBand, TQRDBText, TQRLabel, TQRMemo, TQRPreview e TQRSysData,
TRadioButton, TRichEdit, TScrollBox, TStringGrid, TTreeView
```

Durante a execução do aplicativo:

```
TBrush, TFont, TPen; TCheckBox, TColorDialog, TComboBox, TDBCheckBox, TDBComboBox, TDBCtrlGrid,
TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupComboBox, TDBLookupList,
TDBLookupListBox, TDBMemo, TDBRadioGroup, TDBText, TDirectoryListBox, TDrawGrid, TDriveComboBox,
TEdit, TFileListBox, , TFrame TForm, TGroupBox, TIWCheckBox, TIWControl, TIWDBCheckBox, TIWDBEdit,
TIWDBFile, T IWDBGrid, TIWDBImage, TIWDBListBox, TIWEdit, TIWFile, TIWLabel, TIWListBox,
TIWLookupCombobox, TLabel, TListBox, TListView, TMaskEdit, TMemo, TNotebook, TOutline, TPageControl,
TPaintBox, TPanel, TQRBand, TQRBand, TQRDBText, TQRLabel, TQRMemo, TQRPreview e TQRSysData,
TRadioButton, TRichEdit, TScrollBox, TStringGrid, TTreeView
```

ColorEachPoint

Descrição

A propriedade ColorEachPoint é uma variável do tipo booleano que determina se cada ponto da série representada pelo objeto deve ser desenhado com uma cor distinta.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChartSeries
```

Durante a execução do aplicativo:

```
TChartSeries
```

ColoredButtons

Descrição

A propriedade ColoredButtons é declarada como uma variável do tipo TButtonSet e consiste num conjunto que define os botões do componente que devem aparecer coloridos.

Exemplo

O trecho de código a seguir faz com que todos os botões apareçam em cores.

```
TMediaPlayer1.ColoredButtons := [btPlay, btPause, btStop, btNext, btPrev, btStep, btBack, btRecord,
btEject]
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMediaPlayer
```

Durante a execução do aplicativo:

```
TMediaPlayer
```

Cols

Descrição

A propriedade Cols é declarada como uma array de listas de strings, onde cada lista armazena as strings das células de uma coluna da grade. O índice da array define o número da coluna a ser acessada, começando com 0. Na realidade, cada coluna é tratada como uma lista de strings. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

A linha de código a seguir adiciona a string 'Nova string' à lista de strings correspondente à terceira coluna de um componente chamado StringGrid1 do tipo TStringGrid:

```
StringGrid1.Cols[2].Add('Nova string');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TStringGrid
```

ColumnMarginInches

Descrição

A propriedade ColumnMarginInches é uma variável do tipo inteiro longo que define, em décimos de polegada, o espaço entre colunas para um relatório de múltiplas colunas.

Componentes aos quais se aplica:

Na fase de projeto:

`TQuickReport`

Durante a execução do aplicativo:

`TQuickReport`

ColumnMarginMM

Descrição

A propriedade ColumnMarginMM é uma variável do tipo inteiro longo que define, em milímetros, o espaço entre colunas para um relatório de múltiplas colunas.

Componentes aos quais se aplica:

Na fase de projeto:

`TQuickReport`

Durante a execução do aplicativo:

`TQuickReport`

Columns

Descrição

Para componentes do tipo TQuickReport, a propriedade Columns é declarada como uma variável do tipo booleana, e define o número de colunas do relatório.

Para componentes dos tipos TDBRadioGroup, TDirectoryListBox, TListBox e TRadioGroup, essa propriedade é declarada como uma variável do tipo inteiro longo (Longint), e indica o número de colunas do controle.

Para controles do tipo TListView, essa propriedade é declarada como uma variável do tipo TListColumns, e define o cabeçalho de uma coluna do componente.

Exemplo

Você pode alterar a propriedade Columns de um componente diretamente no Object Inspector ou durante a execução do aplicativo incluindo uma linha de código como:

```
ListBox1.Columns := 3;
```

Componentes aos quais se aplica:

Na fase de projeto:

`TDBRadioGroup, TDirectoryListBox, TListBox, TListView, TQuickReport e TRadioGroup`

Durante a execução do aplicativo:

`TDBRadioGroup, TDirectoryListBox, TListBox, TListView, TQuickReport e TRadioGroup`

ColWidths

Descrição

A propriedade ColWidths é declarada com uma array de inteiros, onde cada item da array define a largura, em pixels, das células da coluna especificada pelo índice. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe a largura das células da primeira coluna de um componente StringGrid1 do tipo TStringGrid quando o usuário seleciona uma célula com o mouse.

```
procedure TForm1.StringGrid1Click(Sender: TObject);
begin
     Label1.Caption := IntToStr(StringGrid1.ColWidths[0]);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

`TDrawGrid e TStringGrid`

COMMAND

Descrição

Essa propriedade é declarada como uma variável do tipo Word inteira e indica o número do comando passado para o Windows usar na mensagem WM_COMMAND a ser enviada para o formulário quando o usuário seleciona um item de menu. Esse comando só é útil se você quiser manipular diretamente a mensagem WM_COMMAND, o que raramente é necessário com o Delphi. Essa propriedade só pode ser acessada durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

`TMenuItem`

COMMANDTEXT

Descrição

Essa propriedade é definida como uma variável do tipo String, e define o comando a ser executado (geralmente na forma de uma declaração SQL).

Componentes aos quais se aplica:

Na fase de projeto:

`TADOCommand e TSQLDataset`

Durante a execução do aplicativo:

`TADOCommand, TADODataset, TADOTable, TADOQuery, TADOStoredProc e TSQLDataset`

COMMANDTIMEOUT

Descrição

Essa propriedade é definida como uma variável inteira, e define o tempo máximo (em segundos) para a execução de um comando (senão considera-se que o comando não foi executado com sucesso). Seu valor default é igual a 30.

Componentes aos quais se aplica:

Na fase de projeto:

`TADOCommand, TADOConnection, TADODataset`

Durante a execução do aplicativo:

`TADOCommand, TADOConnection, TADODataset, TADOTable, TADOQuery e TADOStoredProc`

COMMANDTYPE

Descrição

Para componentes do tipo TADOCommand e TADODataset, essa propriedade é definida como uma variável do tipo TCommandType, e define o tipo de comando a ser executado.

Para estes componentes, esta propriedade pode assumir um dos seguintes valores:

Valor	Significado
CmdUnknown:	O tipo de comando é desconhecido
cmdText	O comando é uma definição textual de uma declaração SQL ou chamada de uma Stored Procedure
cmdTable	O comando é, na realidade, o nome de uma tabela.
cmdStoredProc	O comando é, na realidade, o nome de uma stored procedure.
cmdFile	O comando se refere ao nome de um arquivo no qual um conjunto de registros está armazenado..
cmdTableDirect	O comando é, na realidade, o nome de uma tabela; sendo todos os campos retornados.

As constantes cmdTable, cmdTableDirect e cmdOpenFile não são usadas com o componente TADOCommand.

Para o componente TSQLDataset, essa propriedade é definida como uma variável do tipo TSQLCommandType, e define o tipo de comando a ser executado.

Para este componente, esta propriedade pode assumir um dos seguintes valores:

Valor	Significado
ctTable	O comando é, na realidade, o nome de uma tabela.
ctStoredProc	O comando é, na realidade, o nome de uma stored procedure.
ctQuery	O comando é uma instrução SQL.

Componentes aos quais se aplica:

Na fase de projeto:

`TADOCommand, TADODataset e TSQLDataset`

Durante a execução do aplicativo:

`TADOCommand, TADODataset, TADOTable, TADOQuery, TADOStoredProc e TSQLDataset`

COMMONAVI

Descrição

Essa propriedade é declarada como uma variável booleana, e define se o clipe a ser exibido pelo componente corresponde a uma das animações armazenadas no arquivo Shell32.dll.

Componentes aos quais se aplica:

Na fase de projeto:

```
TAnimate
```

Durante a execução do aplicativo:

```
TAnimate
```

ComponentCount

Descrição

A propriedade ComponentCount é declarada como uma variável inteira e indica o número de componentes possuídos pelo componente atual, e listados na sua propriedade Components. Essa propriedade só pode ser acessada durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Crie um formulário em branco (chamado Form1) e inclua um componente TLabel (chamado Label1). Se durante a execução do aplicativo você quiser exibir o valor da propriedade ComponentCount em Label1, inclua as seguintes linhas de código:

```
Label1.Caption := IntToStr(Form1.ComponentCount);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos.

ComponentIndex

Descrição

A propriedade ComponentIndex é declarada como uma variável do tipo inteiro e indica a posição do componente na lista do seu componente-proprietário. O primeiro componente da lista possui o valor da propriedade ComponentIndex igual a 0, o segundo possui o valor 1 e assim por diante. Essa propriedade só pode ser acessada durante a execução do aplicativo e, ao contrário do que consta no Help On-line do Delphi, pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Crie um formulário em branco (chamado Form1) e inclua um componente TButton (chamado Button1), um componente TPanel (chamado Panel1) e um componente TRadioButton (chamado TRadioButton1).

Se durante a execução do aplicativo você quiser colocar o valor da propriedade ComponentIndex de cada um dos componentes acima em três variáveis inteiras a, b e c, inclua as seguintes linhas de código:

```
a := Form1.Button1.ComponentIndex;
b := Form1.Panel1.ComponentIndex;
c := Form1.RadioButton11.ComponentIndex;
```

Para alterar a propriedade ComponentIndex de um componente, basta incluir uma linha de código do tipo:

```
Form1.Panel1.ComponentIndex := a;
```

Nesse caso, o Delphi se encarrega de redefinir a propriedade ComponentIndex dos demais controles, de modo que não ocorra duplicação de índice.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos.

Components

Descrição

A propriedade Components é declarada como uma array que contém todos os componentes que pertencem ao componente atual. Essa propriedade só pode ser acessada durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Observação: Não confundir a propriedade *Components* com a propriedade *Controls*. A propriedade *Components* contém todos os componentes que pertencem ao componente atual, enquanto a propriedade *Controls* contém todos os controles que pertencem ao componente atual, no caso de este ser um controle (lembre-se de que todo controle é um componente mas nem todo componente é um controle).

Exemplo

Se você quiser que um componente chamado Label1 tenha a sua propriedade Caption igual à do terceiro componente da propriedade *Components* de um formulário chamado Form1, basta digitar a seguinte linha de código:

```
Label1.Caption := TButton(Form1.Components[2]).Caption;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos.

ConfirmDelete

Descrição

A propriedade ConfirmDelete é declarada como uma variável booleana e determina se deve ser exibida uma mensagem de confirmação quando o usuário tentar deletar um registro do banco de dados com o controle TDBNavigator.

Exemplo

Você pode alterar a propriedade ConfirmDelete diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
DBNavigator1.ConfirmDelete := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBNavigator
```

Durante a execução do aplicativo:

```
TDBNavigator
```

Connected

Descrição

A propriedade Connected é uma variável booleana que define se o componente está ou não conectado a um banco de dados.

Exemplo

Você pode alterar a propriedade Connected de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
Database1.Connected := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TSQLConnection, TADOConnection, TIBDatabase, Tdatabase, TRDSConnection
```

Durante a execução do aplicativo:

```
TSQLConnection, TADOConnection, TDatabase
```

CONNECTION

Descrição

Para componentes de acesso via ADO, essa propriedade é definida como uma variável do tipo TADOConnection, e define o tipo de conexão que será usada pelo componente.

Para componentes de acesso via DBExpress (TSQLConnection), essa propriedade é definida como uma variável do tipo string, e identifica a conexão representada pelo componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TSQLConnection, TADOCommand, TADODataset, TADOTable, TADOQuery  e TADOStoredProc
```

Durante a execução do aplicativo:

```
TSQLConnection, TADOCommand, TADODataset, TADOTable, TADOQuery  e TADOStoredProc
```

CONNECTIONSTRING

Descrição

Essa propriedade é definida como uma variável do tipo WideString e define como será feita a conexão ao banco de dados.

As propriedades Connection e ConnectionString são mutuamente excludentes.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOCommand, TADOConnection, TADODataset, TADOTable, TADOQuery  e TADOStoredProc
```

Durante a execução do aplicativo:

```
TADOCommand, TADOConnection, TADODataset, TADOTable, TADOQuery  e TADOStoredProc
```

CONNECTMODE

Descrição

A propriedade ConnectMode é declarada como uma variável do tipo TDataMode que estabelece como será feita a ligação com a aplicação servidora.

Tabela de Valores:

Valor	Significado
ddeAutomatic	A ligação é feita automaticamente quando o formulário que contém o componente TDDEClientConv é criado, durante a execução do aplicativo.
ddeManual	A ligação é feita através do método OpenLink, durante a execução do aplicativo.

Exemplo

Você pode alterar o valor da propriedade ConnectMode de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
DdeClientConv1.ConnectMode := ddeManual;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDDEClientConv
```

Durante a execução do aplicativo:

```
TDDEClientConv
```

CONTEXT

Descrição

A propriedade Context é declarada como uma variável do tipo string, e contém informações sobre o erro que gerou a exceção.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
EReconcileError, EUpdateError
```

CONTROLCOUNT

Descrição

A propriedade ControlCount é declarada como uma variável inteira e indica o número de controles pertencentes ao controle atual. Essa propriedade só pode ser acessada durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Se você quiser saber quantos controles pertencem a um componente Panel (chamado Panel1) de um formulário chamado Form1 e atribuir esse valor a uma variável inteira *a*, basta digitar a seguinte linha de código:

```
a := Form1.Panel1.ControlCount;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os controles.

CONTROLS

Descrição

A propriedade Controls é declarada como uma array que contém todos os controles que pertencem ao controle atual. Essa propriedade só pode ser acessada durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Não confundir a propriedade Controls com a propriedade Components. A propriedade Components contém todos os componentes que pertencem ao componente atual, enquanto a propriedade Controls contém todos os controles que pertencem ao controle atual (lembre-se de que todo controle é um componente mas nem todo componente é um controle).

Exemplo

Se você quiser que um componente chamado Label1 tenha a sua propriedade Caption igual à do terceiro elemento da propriedade Controls de um formulário chamado Form1, basta digitar a seguinte linha de código:

```
Label1.Caption := TButton(Form1.Controls[2]).Caption;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os controles.

CONTROLTYPE

Descrição

Essa propriedade é declarada como uma variável do tipo TDecisionControlType e responde a alterações no componente DecisionPivot.

Essa propriedade pode receber um dos valores mostrados na tabela a seguir.

Valor	Significado
xtCheck	A seleção de um dos botões correspondentes às dimensões manipuladas pelo componente abre ou fecha a dimensão associada ao botão.
xtRadio	A seleção de um dos botões correspondentes às dimensões manipuladas pelo componente abre ou fecha a dimensão associada ao botão e fecha as demais dimensões associadas àquela direção.
xtRadionEx	A seleção de um dos botões correspondentes às dimensões manipuladas pelo componente abre ou fecha a dimensão associada ao botão e fecha ou abre outra dimensão associada àquela direção.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDecisionSource
```

Durante a execução do aplicativo:

```
TDecisionSource
```

ConvertDlgHelp

Descrição

A propriedade ConvertDlgHelp é uma variável do tipo THelpContext que fornece um número ao qual está associado um tópico de auxílio sensível ao contexto para a caixa de diálogo Convert. A caixa de diálogo Convert permite que o usuário converta um objeto OLE em outro tipo de objeto.

Exemplo

Você pode definir o valor da propriedade ConvertDlgHelp diretamente no Object Inspector ou mediante uma linha de código, como:

```
OLEContainer1.ConvertDlgHelp := 8;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOLEContainer
```

Durante a execução do aplicativo:

```
TOLEContainer
```

Copies

Descrição

A propriedade Copies é uma variável inteira que define o número de cópias a serem

impressas.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
PrintDialog1.Copies := 1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPrintDialog, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TPrintDialog, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

Count

Descrição

Para componentes do tipo TIndexDefs, essa propriedade é declarada como uma variável inteira e define o número de elementos armazenados na propriedade Items.

Para componentes do tipo TFieldDefs, essa propriedade é declarada como uma variável inteira e define o número de objetos armazenados.

Para componentes dos tipos TList, TStringList, Tstrings e TMenuItem, essa propriedade é declarada como uma variável inteira e define o número de itens armazenados em uma lista ou item de menu.

Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIndexDefs, TFieldDefs, TList, TStringList, Tstrings e TMenuItem
```

CTL3D

Descrição

A propriedade Ctl3D é declarada como uma variável booleana e determina se o controle terá um aspecto tridimensional (True) ou bidimensional (False).

Exemplo

Você pode alterar a propriedade Ctl3D de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
RadioButton1.Ctl3D := False;
```

Observação: Para que a propriedade Ctl3D funcione com componentes dos tipos TRadioButton, TCheckBox e diálogos comuns do Windows, o arquivo CTL3DV2.DLL deve estar presente no path do sistema.

Componentes aos quais se aplica:

Na fase de projeto:

```
TCheckBox, TColorDialog, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage,
TDBListBox, TDBLookupCombo, TDBLookupComboBox, TDBLookupList, TDBLookupListBox, TDBMemo,
TDBNavigator, TDBRadioGroup, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox,
TFindDialog, TFilterComboBox, TFontDialog, TFrame, TForm, TGroupBox, TListBox, TListView, TMaskEdit,
TMemo, TNotebook, TOLEContainer, TOpenDialog, TOpenPictureDialog, TOutline, TPanel, TQRPreview,
TRadioButton, TReplaceDialog, TRichEdit, TSaveDialog, TSavePictureDialog, TScrollBar, TScrollBox,
TStringGrid, TTrackBar e TTreeView
```

Durante a execução do aplicativo:

```
TCheckBox, TColorDialog, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage,
TDBListBox, TDBLookupCombo, TDBLookupComboBox, TDBLookupList, TDBLookupListBox, TDBMemo,
TDBNavigator, TDBRadioGroup, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox,
TFindDialog, TFilterComboBox, TFontDialog, ,TFrame ,  TForm, TGroupBox, TListBox, TListView,
TMaskEdit, TMemo, TNotebook, TOLEContainer, TOpenDialog, TOpenPictureDialog, TOutline, TPanel,
TQRPreview, TRadioButton, TReplaceDialog, TRichEdit, TSaveDialog, TSavePictureDialog, TScrollBar,
TScrollBox, TStringGrid, TTrackBar e TTreeView
```

CURRENCY

Descrição

A propriedade Currency é uma variável booleana usada para controlar o formato de exibição do campo. Quando Currency é igual a True a formatação é feita com a função FloatToText usando ffCurrency como parâmetro para exibir texto ou ffFixed como parâmetro para editar texto. Quando Currency é igual a False, a formatação é feita com a função FloatToTextFmt. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar a propriedade Currency de um componente através de uma linha de código, como:

```
FloatField1.Currency := True;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBCDField, TCurrencyField e TFloatField
```

CurrentMemory

Descrição

Esta propriedade é declarada como uma variável do tipo inteiro longo, e define a quantidade de memória do servidor usada pelo banco de dados.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDatabaseInfo
```

Durante a execução do aplicativo:

```
TIBDatabaseInfo
```

CurrentPage

Descrição

A propriedade Currentpage é uma variável inteira que define o número da página corrente do relatório.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

CurrentSum

Descrição

Essa propriedade é declarada como uma variável inteira, e define o sumário corrente do componente DecisionCube ao qual está vinculado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDecisionSource
```

Durante a execução do aplicativo:

```
TDecisionSource
```

Cursor

Descrição

A propriedade Cursor é declarada como uma variável do tipo TCursor e indica a imagem exibida pelo ponteiro do mouse quando este passa sobre o controle.

Tabela de Valores:

Valor	Significado
crDefault	Ponteiro do mouse em forma de seta inclinada.
crArrow	Ponteiro do mouse em forma de seta inclinada.
crCross	Ponteiro do mouse em forma de cruz.
crIBeam	Ponteiro do mouse em forma de "I".
crSize	Ponteiro do mouse com quatro setas.
crSizeNESW	Ponteiro do mouse com duas setas na direção NE-SW.

continua

continuação

Valor	Significado
crSizeNS	Ponteiro do mouse com duas setas na direção N-S.
crSizeNWSE	Ponteiro do mouse com duas setas na direção NW-SE.
crSizeWE	Ponteiro do mouse com duas setas na direção W-E.
crUpArrow	Ponteiro do mouse em forma de seta apontando para cima.
crHourglass	Ponteiro do mouse em forma de ampulheta.
crDrag	Ponteiro do mouse em forma de símbolo de arraste.
crNoDrop	Ponteiro do mouse indicando que um componente não pode ser solto sobre outro.
crHSplit	Ponteiro do mouse em forma de dupla seta horizontal.
crVSplit	Ponteiro do mouse em forma de dupla seta vertical.

Exemplo

Se você quiser que o ponteiro do mouse tenha o formato de uma cruz ao passar sobre um componente Shape1 do tipo TShape, basta definir a sua propriedade Cursor como crCross no Object Inspector.

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles, objetos da classe TChartSeries.

Durante a execução do aplicativo:

Todos os controles, objetos da classe TChartSeries.

Cursors

Descrição

Para um componente TScreen, a propriedade Cursors é declarada como uma variável do tipo HCursor e dá acesso a uma lista de cursores disponíveis para a sua aplicação. Essa propriedade só está disponível durante a execução do aplicativo.

Tabela de Valores:

Valor	Índice	Significado
crDefault	0	Ponteiro do mouse em forma de seta inclinada.
crNone	-1	Invisível.
crArrow	-2	Ponteiro do mouse em forma de seta inclinada.
crCross	-3	Ponteiro do mouse em forma de cruz.
crIBeam	-4	Ponteiro do mouse em forma de "I".
crSize	-6	Ponteiro do mouse com quatro setas.
crSizeNESW	-7	Ponteiro do mouse com duas setas na direção NE-SW.
crSizeNS	-8	Ponteiro do mouse com duas setas na direção N-S.
crSizeNWSE	-9	Ponteiro do mouse com duas setas na direção NW-SE.
crSizeWE	-8	Ponteiro do mouse com duas setas na direção W-E.

continua

continuação

Valor	Índice	Significado
crUpArrow	-10	Ponteiro do mouse em forma de seta apontando para cima.
crHourGlass	-11	Ponteiro do mouse em forma de ampulheta.
crDrag	-12	Ponteiro do mouse em forma de símbolo de arraste.
crNoDrop	-13	Ponteiro do mouse indicando que um componente não pode ser solto sobre outro.
crHSplit	-14	Ponteiro do mouse em forma de dupla seta horizontal.
crVSplit	-15	Ponteiro do mouse em forma de dupla seta vertical.
crMultiDrag	-16	Ponteiro do mouse em forma de arraste múltiplo.

Exemplo

Se você possuir um ícone com nome MyCursor e quiser torná-lo a imagem do ponteiro do mouse ao passar sobre um formulário, inclua o seguinte trecho de código:

```
const
     crMyCursor = 5;
procedure TForm1.FormCreate(Sender: TObject);
begin
     Screen.Cursors[crMyCursor] := LoadCursor(1, 'NewCursor');
     Cursor := crMyCursor;
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TScreen
```

CursorXPos

Descrição

A propriedade CursorXPos é uma variável inteira que define a posição horizontal do texto, em unidades de impressão (pontos).

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

CursorYPos

Descrição

A propriedade CursorYPos é uma variável inteira que define a posição vertical do texto, em unidades de impressão (pontos).

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

CUSTOMCOLORS

Descrição

A propriedade CustomColors, que é declarada como uma variável do tipo TStrings, armazena uma lista de strings e especifica as cores personalizadas que estarão disponíveis na caixa de diálogo.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector (com o string list editor) ou mediante a inclusão de um trecho de código, como:

```
if ColorDialog1.Execute then ListBox1.Items.AddStrings(ColorDialog1.CustomColor
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TColorDialog
```

Durante a execução do aplicativo:

```
TColorDialog
```

DATA

Descrição

A propriedade Data é declarada como uma variável do tipo Pointer e especifica qualquer dado que você queira associar com um item de um componente do tipo TOutline. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TOutlineNode
```

DATABASE

Descrição

A propriedade Database é declarada como uma variável do tipo TDataSet que define o componente de banco de dados associado. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBDatabaseinfo, TIBEvents, TIBSQL, TTable e TQuery, TDecisionQuery
```

DATABASECOUNT

Descrição

A propriedade DatabaseCount é declarada como uma variável inteira e especifica o número de componentes TDataBase (no caso de TSession) ou TIBDatabase (no caso de TIBTransaction) existentes na aplicação. Essa propriedade só está definida durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBTransaction, TSession
```

DatabaseName

Descrição

A propriedade Database é declarada como uma variável do tipo string e define o nome ou alias do banco de dados acessado pelo componente.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector (com o string list editor) ou mediante a inclusão de um trecho de código, como:

```
Table1.DatabaseName := 'Banco_de_Dados';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDataBase, TIBDataBase, TIBTable, TIBQuery, TIBStoredProc
TTable, TQuery, TDecisionQuery e TStoredProc
```

Durante a execução do aplicativo:

```
TDataBase, TTable, TQuery, TDecisionQuery e TStoredProc
```

Databases

Descrição

A propriedade DataBases é declarada como uma array de objetos do tipo TDataBase e mantém uma lista dos objetos do tipo TDataBase ou TIBDatabase disponíveis em uma aplicação. Essa propriedade só está definida durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir fecha todos os bancos de dados associados a uma aplicação.

```
with Session do
   while DatabaseCount <> 0 do
     Databases[0].Close;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBTransaction, TSession
```

DataField

Descrição

A propriedade DataField é declarada como uma variável do tipo string e indica o campo a que esse controle corresponde no registro de um banco de dados.

Exemplo

Você pode definir a propriedade DataField diretamente no Object Inspector ou através de uma linha de código como:

```
DBEdit1.DataField := 'NOME';
```

que define a propriedade DataField de um componente chamado DBEdit1 do tipo TDBEdit como sendo igual a 'NOME'.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBCheckBox, TDBComboBox, TDBEdit, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupComboBox,
TDBLookupList, TDBLookupListBox, TDBMemo, TDBRadioGroup, TDBText, TIWDBCheckBox, TIWDBComboBox,
TIWDBEdit, TIWDBImage, TIWDBListBox, TIWDBLookupCombo, TIWDBLookupComboBox, TIWDBLookupListBox,
TIWDBMemo, TIWDBText, TQRCustomControl, TQRDBCalc, TQRDBText e TQRGroup
```

Durante a execução do aplicativo:

```
TDBCheckBox, TDBComboBox, TDBEdit, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupComboBox,
TDBLookupList, TDBLookupListBox, TDBMemo, TDBRadioGroup, TDBText, TIWDBCheckBox, TIWDBComboBox,
TIWDBEdit, TIWDBImage, TIWDBListBox, TIWDBLookupCombo, TIWDBLookupComboBox, TIWDBLookupListBox,
TIWDBMemo, TIWDBText,TQRCustomControl, TQRDBCalc, TQRDBText e TQRGroup
```

DataSet

Descrição

Para componentes dos tipos TDataSource, TIBUpdateSQL e TUpdateSQL, essa propriedade é declarada como uma variável do tipo TDataSet e indica o componente que representa um banco de dados ao qual o componente está associado.

Para componentes dos tipos TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField, TDecisionCube, TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TRvDatasetConnection, TSmallintField, TStringField, TTimeField, TVarBytesField, TUpdateSQL e TWordField essa propriedade é declarada como uma variável do tipo TDataSet e indica o banco de dados ao qual o campo se refere. Para esses componentes, essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar o valor da propriedade DataSet mediante uma linha de código como a seguir, onde WordField é uma variável do tipo TWordField.

```
WordField.DataSet:= Banco_de_Dados
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDecisionCube, TDataSource, TRvDatasetConnection
```

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDataSource,
TDateField, TDateTimeField, TFloatField, TDecisionCube, TGraphicField, TIntegerField, TMemoField,
TRvDatasetConnection, TSmallintField, TStringField, TTimeField, TVarBytesField, TUpdateSQL e
TWordField
```

DataSetCount

Descrição

A propriedade DataSetCount é declarada como uma variável inteira que define quantos componentes dos tipos TIBDataset, TIBTable, TIBQuery, TIBStoredProc, TTable, TQuery, TDecisionQuery e TStoredProc estão acessando o banco de dados representado por esse componente. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBDatabase, TDatabase
```

DataSets

Descrição

A propriedade DataSets é declarada como uma array de componentes derivados de TDBDataSet que define os componentes dos tipos TTable, TQuery, TDecisionQuery e TStoredProc que estão acessando o banco de dados representado por esse componente.

Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADOConnection, TIBDatabase, Tdatabase, TRDSConnection
```

DataSize

Descrição

A propriedade DataSize é declarada como uma variável inteira e define o número de bytes ocupados pelo campo na memória. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir exibe uma mensagem informando o número de bytes ocupado por WordField, que é uma variável do tipo TWordField.

```
ShowMessage(IntToStr(WordField.DataSize);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField, TStringField,
TTimeField, TVarBytesField e TWordField
```

DataSource

Descrição

Para componentes dos tipos TDBCheckBox, TDBComboBox, TDBEdit, TDBCtrlGrid, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDBNavigator, TDBRadioGroup, TIWDBCheckBox, TIWDBComboBox, TIWDBEdit, TIWDBGrid, TIWDBImage, TIWDBListBox, TIWDBLookupCombo, IWDBLookupList, TIWDBMemo, TIWDBNavigator, TQuickReport, TQRDetailLink e TDBText, a propriedade DataSource é declarada como uma variável do tipo TDataSource e indica o componente ao qual está vinculado o banco de dados cujos campos se quer exibir.

Para componentes dos tipos TADOCommand, TADODataset, TADOTable, TADOQuery, e TADOStoredProc, TQuery, TDecisionQuery e TClientDataSet a propriedade DataSource é declarada como uma variável do tipo TDataSource e indica o componente ao qual está vinculado o banco de dados ao qual se pode atribuir valores não limitados aos definidos por sua propriedades Params ou seu método ParamByName.

Para objetos da classe TChartSeries, determina a fonte de dados que definem os pontos da série.

Exemplo

Você pode definir a propriedade DataSource diretamente no Object Inspector ou através de uma linha de código, como:

```
DBEdit1.DataSource:= DataSource1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TChartSeries, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBCtrlGrid, TDBImage, TDBListBox,
TDBLookupCombo, TDBLookupComboBox, TDBLookupList, TDBLookupListBox, TDBMemo, TDBNavigator,
TDBRadioGroup, TDBText, TIWDBCheckBox, TIWDBComboBox, TIWDBEdit, TIWDBGrid, TIWDBImage,
TIWDBListBox, TIWDBLookupCombo,  IWDBLookupList, TIWDBMemo, TIWDBNavigator, TQRCustomControl,
```

```
TQRDBCalc, TQRDBText, TQRDetailLink, TQRGroup, TQuickReport, TQRDetailLink TClientDataSet, TQuery,
TdecisionQuery, TADOCommand, TADODataset, TADOTable, TADOQuery, e TADOStoredProc
```

Durante a execução do aplicativo:

```
TChartSeries, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBCtrlGrid, TDBImage, TDBListBox,
TDBLookupCombo, TDBLookupComboBox, TDBLookupList, TDBLookupListBox, TDBMemo, TDBNavigator,
TDBRadioGroup, TDBText, TIWDBCheckBox, TIWDBComboBox, TIWDBEdit, TIWDBGrid, TIWDBImage,
TIWDBListBox, TIWDBLookupCombo,  IWDBLookupList, TIWDBMemo, TIWDBNavigator, TQRCustomControl,
TQRDBCalc, TQRDBText, TQRDetailLink, TQRGroup, TQuickReport, TQRDetailLink TClientDataSet, TQuery,
TDecisionQuery TADOCommand, TADODataset, TADOTable, TADOQuery, e TADOStoredProc
```

DATATYPE

Descrição

A propriedade DataType é declarada como do tipo TFieldType e define o tipo do campo.

Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFieldDef, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

DBFILENAME

Descrição

Esta propriedade é declarada como uma variável do tipo string, e define o nome do arquivo no qual o banco de dados está armazenado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDatabaseInfo
```

Durante a execução do aplicativo:

```
TIBDatabaseInfo
```

DBHANDLE

Descrição

A propriedade DBHandle é declarada como uma variável do tipo HDBIDB, que permite acesso direto às funções da API do Borland Database Engine (BDE). Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TTable, TQuery, TDecisionQuery e TStoredProc
```

DBLOCALE

Descrição

A propriedade DBLocale é declarada como uma variável do tipo TLocale, que permite acesso direto ao Borland Database Engine (BDE). Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TTable, TQuery, TDecisionQuery e TStoredProc
```

DBSQLDialect

Descrição

Esta propriedade é declarada como uma variável do tipo inteiro e define o tipo de dialeto SQL usado pelo banco de dados.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBDatabaseinfo, TIBDataBase
```

DDEConv

Descrição

A propriedade DDEConv é declarada como uma variável do tipo TDDEClientConv e define o componente do tipo TDDEClientConv ao qual esse item está associado em uma conversação DDE.

Exemplo

Você pode alterar o valor da propriedade DDEConv de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
DDEClientItem1.DDEConv := DDEClientConv1
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDDEClientConv
```

Durante a execução do aplicativo:

```
TDDEClientConv
```

DDEItem

Descrição

A propriedade DDEItem é declarada como uma variável do tipo string e define o item de uma conversação DDE.

Exemplo

Você pode alterar o valor da propriedade DDEItem de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
DDEClientItem1.DDEItem := 'DDEC1';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDDEClientConv
```

Durante a execução do aplicativo:

```
TDDEClientConv
```

DDEService

Descrição

A propriedade DDEService é declarada como uma variável do tipo string que define o nome da aplicação servidora.

Exemplo

Você pode alterar o valor da propriedade DDEService de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
DdeClientConv1.DDEService := 'Programa';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDDEClientConv
```

Durante a execução do aplicativo:

```
TDDEClientConv
```

DDETopic

Descrição

A propriedade DDETopic é declarada como uma variável do tipo string que define o tópico da conversação DDE.

Exemplo

Você pode alterar o valor da propriedade DDETopic de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
DdeClientConv1.DDETopic := 'Topico';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDDEClientConv
```

Durante a execução do aplicativo:

```
TDDEClientConv
```

DecisionCube

Descrição

Essa propriedade define o componente DecisionCube associado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDecisionSource
```

Durante a execução do aplicativo:

```
TDecisionSource
```

DecisionSource

Descrição

Essa propriedade define o componente DecisionSource associado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDecisionGrid, TDecisionGraph, TDecisionPivot
```

Durante a execução do aplicativo:

```
TDecisionGrid, TDecisionGraph, TDecisionPivot
```

Default

Descrição

A propriedade Default é declarada como uma variável do tipo booleano e indica se um controle do tipo TButton ou TBitBtn associa o seu evento OnClick ao pressionamento da tecla Enter.

Exemplo

Coloque dois botões chamados Button1 e Button2 num formulário chamado Form1 e defina a propriedade Default de Button1 como True no Object Inspector. No evento OnClick de Button1, inclua a seguinte linha de código:

```
if Form1.Color := clYellow then Form1.Color := clRed else Form1.Color := clYellow;
```

Execute o aplicativo, pressione seguidamente a tecla Enter e você verá a cor do formulário alternar entre vermelho e amarelo, a menos que Button2 receba o foco da aplicação.

Se durante a execução do aplicativo um outro botão chamado Button2 obtiver o Foco, passará temporariamente a ser o botão default, até que o foco seja dirigido para um componente que não seja um botão, quando Button1 voltará a ser o botão Default.

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn e TButton
```

Durante a execução do aplicativo:

```
TBitBtn e TButton
```

DefaultColWidth

Descrição

A propriedade DefaultColWidth é uma variável inteira que especifica a largura de todas as colunas do componente. Se você quiser alterar a largura de uma única coluna, redefina a propriedade ColWidths durante a execução do aplicativo.

Exemplo

Você pode alterar a propriedade DefaultColWidth de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
DrawGrid1.DefaultColWidth := 50;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDrawGrid, TDecisionGrid e TStringGrid
```

Durante a execução do aplicativo:

```
TDrawGrid, TDecisionGrid e TStringGrid
```

DefaultDatabase

Descrição

Esta propriedade é declarada como um objeto da classe TIBTransaction, e define o componente default que representa um banco de dados associado a transação.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBTransaction
```

Durante a execução do aplicativo:

```
TIBTransaction
```

DefaultDrawing

Descrição

A propriedade DefaultDrawing é uma variável booleana que determina se uma célula deve ser pintada e o item que ela contém deve ser desenhado automaticamente. Se o valor da variável for False, o desenho da célula deve ser programado nos eventos OnDrawCell e OnDrawDataCell.

Exemplo

Você pode alterar a propriedade DefaultDrawing de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
DBGrid1.DefaultDrawing := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBGrid, TDrawGrid e TString
```

Durante a execução do aplicativo:

```
TDBGrid, TDrawGrid e TString
```

DefaultExt

Descrição

A propriedade DefaultExt é declarada como uma variável do tipo TFileExt que especifica a extensão a ser adicionada ao nome de um arquivo quando o usuário digita o nome de um arquivo sem a sua extensão.

Exemplo

Você pode alterar a propriedade DefaultExt de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
OpenDialog1.DefaultExt := 'TXT';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog
```

Durante a execução do aplicativo:

```
TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog
```

DefaultIndex

Descrição

A propriedade DefaultIndex é declarada como uma variável booleana, e define se os registros da tabela representada pelo componente devem ser ordenados por um índice default quando a tabela é aberta.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBTable, TTable
```

Durante a execução do aplicativo:

```
TTable
```

DefaultRowHeight

Descrição

A propriedade DefaultRowHeight é uma variável inteira que especifica a altura de todas as linhas do componente. Se você quiser alterar a altura de uma única linha, redefina a propriedade RowHeigths durante a execução do aplicativo.

Exemplo

Você pode alterar a propriedade DefaultRowHeigth de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
DrawGrid1. DefaultRowHeigth := 50;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDrawGrid e TStringGrid
```

Durante a execução do aplicativo:

```
TDrawGrid e TStringGrid
```

DefaultTransaction

Descrição

A propriedade DefaultTransaction é declarada como um objeto da classe TIBTransaction e define a transação default associada ao banco de dados representado pelo componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataBase
```

Durante a execução do aplicativo:

```
TIBDataBase
```

DeleteSQL

Descrição

Essa propriedade armazena uma lista de strings contendo a declaração SQL usada para deletar um registro.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataset, TIBUpdateSQL, TUpdateSQL
```

Durante a execução do aplicativo:

```
TIBDataset, TIBUpdateSQL, TUpdateSQL
```

DESTINATION

Descrição

A propriedade Destination é declarada como uma variável do tipo TTable que define o componente ao qual está associada uma tabela de um banco de dados à qual serão adicionados os registros operados pelo método Execute de um componente do tipo TBatchMove.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
BatchMove1.Destination := Table1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBatchMove
```

Durante a execução do aplicativo:

```
TBatchMove
```

DEVICE

Descrição

A propriedade Device é declarada como uma variável do tipo TFontDialogDevice e define o(s) dispositivo(s) afetado(s) pela fonte selecionada.

Tabela de Valores:

Valor	Significado
fdScreen	Afeta a tela.
fdPrinter	Afeta a impressora.
fdBoth	Afeta a tela e a impressora.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
FontDialog1.Device := fdBoth;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TFontDialog
```

Durante a execução do aplicativo:

```
TFontDialog
```

DeviceID

Descrição

A propriedade DeviceID é declarada como uma variável do tipo Word e define uma ID para o dispositivo. Essa propriedade só está definida durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente Edit1 do tipo TEdit exiba a ID do dispositivo multimídia corrente:

```
Edit1.Text := IntToStr(MediaPlayer1.DeviceID);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

DeviceType

Descrição

A propriedade DeviceType é declarada como uma variável do tipo TMPDeviceTypes e define o tipo de dispositivo a ser aberto com o método Open. Os valores possíveis para essa propriedade são dtAutoSelect, dtAVIVideo, dtCDAudio, dtDAT, dtDigitalVideo, dtMMMovie, dtOther, dtOverlay, dtScanner, dtSequencer, dtVCR, dtVideodisc e dtWaveAudio.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
MediaPlayer1.DeviceType := dtDat;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMediaPlayer
```

Durante a execução do aplicativo:

```
TMediaPlayer
```

DimensionMap

Descrição

Essa propriedade é declarada como um objeto da classe TcubeDims e define as dimensões e sumários manipulados pelo componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDecisionSource
```

Durante a execução do aplicativo:

```
TDecisionSource
```

Directory

Descrição

A propriedade Directory é declarada como uma variável do tipo string e define o diretório apontado pelo componente.

Exemplo

O trecho de código a seguir faz com que um componente chamado FileListBox1 do tipo TFileListBox exiba os arquivos existentes no diretório apontado pelo componente DirectoryListBox1 do tipo TDirectoryListBox.

```
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
    FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDirectoryListBox e TFileListBox
```

DirLabel

Descrição

A propriedade DirLabel é declarada como uma variável do tipo TLabel, que permite exibir o diretório corrente como a propriedade Caption de um componente do tipo TLabel.

Exemplo

Coloque um componente chamado Label1 do tipo TLabel e um componente chamado DirectoryListBox1 do tipo TDirectoryListBox em um formulário. Defina a propriedade DirLabel de DirectoryListBox1 como Label1 diretamente no Object Inspector ou mediante uma linha de código, como:

```
DirectoryListBox1.DirLabel := Label1;
```

Isto faz com que Label1 exiba sempre o diretório apontado por DirectoryListBox1.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDirectoryListBox
```

Durante a execução do aplicativo:

```
TDirectoryListBox
```

DirList

Descrição

A propriedade FileList é declarada como uma variável do tipo TDirectoryListBox e serve para fazer a conexão com um componente do tipo TDirectoryListBox. Ao fazer esse tipo de conexão, o componente TDirectoryListBox apontado por DirList exibirá sempre o diretório atual do drive corrente.

Exemplo

Coloque um componente chamado DriveComboBox1 do tipo TDriveComboBox e um componente chamado DirectoryListBox1 do tipo TDirectoryListBox em um Formulário.

Defina a propriedade DirList de DriveComboBox 1 como DirectoryListBox1 diretamente no Object Inspector ou mediante uma linha de código como:

```
DriveComboBox 1.DirList := DirectoryListBox 1;
```

Isto faz com que DirectoryListBox1 exiba sempre os diretórios do drive apontado por DriveComboBox 1.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDriveComboBox
```

Durante a execução do aplicativo:

```
TDriveComboBox
```

DisableIfNoHandler

Descrição

A propriedade DisableIfNoHandler é declarada como uma variável booleana, e define se os controles e itens de menu associados ao objeto devem ser desabilitados caso não exista nenhum procedimento associado a evento definido para esse objeto.

Componentes aos quais se aplica:

Na fase de projeto:

```
TAction
```

Durante a execução do aplicativo:

```
TAction
```

Display

Descrição

A propriedade Display é declarada como uma variável do tipo TWinControl e define o controle a ser usado para exibição dos resultados durante a execução de um dispositivo multimídia.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
MediaPlayer1.Display := Form1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMediaPlayer
```

Durante a execução do aplicativo:

```
TMediaPlayer
```

DisplayFormat

Descrição

A propriedade DisplayFormat é declarada como uma variável do tipo string e define o formato de exibição de um campo. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TDateField, TDateTimeField, TIntegerField, TSmallintField, TTimeField e TWordField
```

DisplayLabel

Descrição

A propriedade DisplayLabel é declarada como uma variável do tipo string e define o cabeçalho a ser exibido na coluna que exibe o valor de um campo num componente do tipo TDBGrid. Se seu valor for igual a Null, o valor armazenado na propriedade FieldName será usado. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar o valor dessa propriedade mediante a inclusão de uma linha de código, como:

```
WordField1.DisplayLabel:= 'coluna';
```

onde WordField1 é uma variável do tipo TWordField.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

DisplayName

Descrição

A propriedade DisplayLabel é declarada como uma variável do tipo Pstring e retorna o nome do campo. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

DisplayPrintDialog

Descrição

Essa propriedade é declarada como uma variável booleana e define se uma caixa de diálogo de impressão deve ser exibida antes de o relatório ser impresso.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou durante a execução do aplicativo, mediante a inclusão de uma linha de código como:

```
QuickReport1.DisplayPrintDialog:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TquickReport
```

Durante a execução do aplicativo:

```
TQuickReport
```

DisplayRect

Descrição

Para controles do tipo TMediaPlayer, a propriedade DisplayRect é declarada como uma variável do tipo TRect, define uma área retangular no controle a ser usado para exibição dos resultados durante a execução de um dispositivo multimídia e só está disponível durante a execução do aplicativo.

Para componentes do tipo TTabControl, essa propriedade é declarada como uma variável do tipo TRect e define a altura, em pixels, da área-cliente de um controle.

Exemplo

Você pode alterar o valor dessa propriedade mediante a inclusão de uma linha de código, como:

```
MediaPlayer1.DisplayRect:= Rect(10, 10, 100, 200);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer e TTabControl
```

DisplayText

Descrição

A propriedade DisplayText é declarada como uma variável do tipo string e retorna o texto exibido em um controle de exibição de campos de bancos de dados (exceto TEdit). Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Você pode exibir o valor dessa propriedade mediante a inclusão de uma linha de código, como:

```
ShowMessage(IntToStr(IntegerField1.DisplayText));
```

onde IntegerField1 é uma variável do tipo TIntegerField.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

DisplayValue

Descrição

A propriedade DisplayValue é declarada como uma variável do tipo string e define a string exibida pelo controle. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que um componente Label1 do tipo TLabel exiba a string armazenada na propriedade DisplayValue de um componente DBLookupCombo1 do tipo TDBLookupCombo:

```
Label1.Caption := DBLookupCombo1.DisplayValue;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBLookupCombo e TDBLookupList
```

DisplayWidth

Descrição

A propriedade DisplayWidth é declarada como uma variável inteira que define o número de caracteres usados para exibir o valor do campo em um controle do tipo TDBGrid. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar o valor dessa propriedade mediante a inclusão de uma linha de código, como:

```
IntegerField1.DisplayWidth:= 15;
```

onde IntegerField1 é uma variável do tipo TIntegerField.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

DitherBackground

Descrição

A propriedade DitherBackground é declarada como uma variável booleana e define se a cor de fundo do componente deve ser ajustada por um processo de Dithering.

Exemplo

Você pode alterar a propriedade DitherBackground de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
TabSet1.DitherBackground := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTabSet
```

Durante a execução do aplicativo:

```
TTabSet
```

DocBackColor

Descrição

A propriedade DocBackColor é declarada como uma variável do tipo TColor e define a cor de fundo default de uma página Web.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
THTML
```

DocForeColor

Descrição

A propriedade DocForeColor é declarada como uma variável do tipo TColor e define a cor default do texto exibido em uma página Web.

Componentes aos quais se aplica:

Na fase de projeto:

```
THTML
```

Durante a execução do aplicativo:

```
THTML
```

DockClientCount

Descrição

A propriedade DockClientCount é declarada como uma variável do tipo inteiro que define o número de controles ancorados no componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TPanel, TChart, TDBChart, TDecisionGraph, TFrame, TForm, TDBCtrlGrid, TDrawGrid, TDecisionGrid
e TStringGrid
```

DockClients

Descrição

Essa propriedade armazena em uma array as referências aos controles ancorados no componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TPanel, TChart, TDBChart, TDecisionGraph, TFrame, TForm, TDBCtrlGrid, TDrawGrid, TDecisionGrid
e TStringGrid
```

DockRect

Descrição

A propriedade DockRect é declarada como uma variável do tipo TRect que define uma área retangular na qual, se liberados, os controles serão ancorados.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TPanel, TChart, TDBChart, TDecisionGraph, TFrame, TForm, TDBCtrlGrid, TDrawGrid, TDecisionGrid
e TStringGrid
```

DOCKSITE

Descrição

A propriedade DockSite é declarada como uma variável do tipo booleano que define se o controle poderá ancorar outros componentes, isto é, se será o alvo de uma operação de drag-e-dock.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TPanel, TChart, TDBChart, TDecisionGraph, TFrame, TForm, TDBCtrlGrid, TDrawGrid, TDecisionGrid
e TStringGrid
```

DOCLINKCOLOR

Descrição

A propriedade DocLinkColor é declarada como uma variável do tipo TColor e define a cor default do texto que representa um link em uma página Web.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
THTML
```

DOCLINKCOLOR

Descrição

A propriedade DocVisitedColor é declarada como uma variável do tipo TColor e define a cor default do texto que representa um link já visitado em uma página Web.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
THTML
```

DOWN

Descrição

A propriedade Down é declarada como uma variável do tipo booleana, e indica se o componente está selecionado (True) ou não (False).

Componentes aos quais se aplica:

Na fase de projeto:

```
TSpeedButton
```

Durante a execução do aplicativo:

```
TSpeedButton
```

DragCursor

Descrição

A propriedade DragCursor é declarada como uma variável do tipo TCursor e indica a imagem exibida pelo ponteiro do mouse quando este passa sobre um controle que pode aceitar o objeto sendo arrastado.

Tabela de Valores:

Consulte a propriedade Cursor.

Exemplo

Se você quiser alterar a propriedade DragCursor de um componente chamado Memo1 do tipo TMemo para crIBeam, basta incluir a seguinte linha de código:

```
Memo1.DragCursor:= crIBeam;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBEdit, TDBGrid, TDBCtrlGrid, TDBImage,
TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDBNavigator, TDBRadioGroup, TDBText,
TDirectoryListBox, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TGroupBox, TImage,
TLabel, TListBox, TMaskEdit, TMemo, .TOutline, TPaintBox, TPanel, TRadioButton, TScrollBar,
TScrollBox, TShape, TDBLookupComboBox, TDBLookupListBox, TTabControl, THeaderControl,
TListView, TPageControl, TStatusBar, TTreeView, TTrackBar, TQRPreview e TNotebook
```

Durante a execução do aplicativo:

```
TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBEdit, TDBGrid, TDBCtrlGrid, TDBImage,
TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDBNavigator, TDBRadioGroup, TDBText,
TDirectoryListBox, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TGroupBox, TImage,
TLabel, TListBox, TMaskEdit, TMemo, .TOutline, TPaintBox, TPanel, TRadioButton, TScrollBar,
TScrollBox, TShape, TDBLookupComboBox, TDBLookupListBox, TTabControl, THeaderControl,
TListView, TPageControl, TStatusBar, TTreeView, TTrackBar, TQRPreview e TNotebook
```

Dragging

Descrição

A propriedade Dragging é declarada como uma variável do tipo booleana que determina se o controle está sendo arrastado. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os controles.

DragMode

Descrição

A propriedade DragMode é declarada como uma variável do tipo TDragMode e define o comportamento de um controle no processo de arrastar e soltar.

Tabela de Valores:

Valor	Significado
dmAutomatic	O controle está pronto para ser arrastado, sem necessidade de programar o início do processo de arraste.
dmManual	O início do processo de arraste deve ser feito por uma chamada ao método BeginDrag.

Exemplo

Se você quiser alterar a propriedade DragMode de um componente chamado Memo1 do tipo TMemo para dmManual, basta incluir a seguinte linha de código:

```
Memo1.DragMode:= dmManual;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid,
TDBCtrlGrid, TDBImage, TDBText, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo,
TDBNavigator, TDBRadioGroup, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit,
TFileListBox, TFilterComboBox, TGroupBox, TImage, TLabel, TListBox, TMaskEdit, TMemo,
TOLEContainer, TOutline, TPaintBox, TRadioButton, TScrollBar, TScrollBox, TShape,
TDBLookupComboBox, TDBLookupListBox, TTabControl, THeaderControl, TListView, TPageControl,
TStatusBar, TTreeView, TTrackBar, TQRPreview, TStringGrid e TNotebook
```

Durante a execução do aplicativo:

```
TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid,
TDBCtrlGrid, TDBImage, TDBText, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo,
TDBNavigator, TDBRadioGroup, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit,
TFileListBox, TFilterComboBox, TGroupBox, TImage, TLabel, TListBox, TMaskEdit, TMemo,
TOLEContainer, TOutline, TPaintBox, TRadioButton, TScrollBar, TScrollBox, TShape,
TDBLookupComboBox, TDBLookupListBox, TTabControl, THeaderControl, TListView, TPageControl,
TStatusBar, TTreeView, TTrackBar, TQRPreview, TStringGrid e TNotebook
```

DRAWINGSTYLE

Descrição

Essa propriedade é declarada como uma variável do tipo TDrawingStyle, e especifica como a imagem será desenhada.

Tabela de Valores:

Valor	Significado
dsFocused	Desenha com uma combinação em 50% da cor do sistema.
dsSelected	Desenha com uma combinação em 25% da cor do sistema.
dsNormal	Desenha a imagem com a cor especificada na propriedade BkColor.
dsTransparent	Desenha com uma máscara, independentemente do valor definido na propriedade BkColor.

Componentes aos quais se aplica:

Na fase de projeto:

```
TImageList
```

Durante a execução do aplicativo:

```
TImageList
```

DRIVE

Descrição

A propriedade Drive é declarada como uma variável do tipo Char que determina o drive exibido em um componente do tipo TDriveComboBox, o drive a que pertence o diretório exibido em um

componente TDirectoryListBox ou o drive a que pertencem os arquivos exibidos em um componente TFileListBox. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode fazer com que um componente DirectoryListBox1 do tipo TDirectoryListBox exiba os diretórios no drive C com a seguinte linha de código:

```
DirectoryListBox1.Drive := 'C';
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDirectoryListBox, TDriveComboBox e TFileListBox
```

DriverName

Descrição

A propriedade DriverName especifica o driver do Borland Database Engine (BDE) ou do DBExpress usado com o banco de dados associado ao componente. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDatabase, TSQLConnection
```

DropDownAlign

Descrição

A propriedade DropDownAlign define o tipo de alinhamento do texto exibido na lista drop-down mostrada pelo componente.

Tabela de Valores:

Valor	Significado
daLeft	Alinhamento à esquerda.
daCenter	Alinhamento pelo centro.
daRight	Alinhamento à direita.

Exemplo

Você pode alterar o valor da propriedade DropDownAlign diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
DBLookupComboBox1.DropDownAlign:= daLeft;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBLookupComboBox
```

Durante a execução do aplicativo:

```
TDBLookupComboBox
```

DropDownCount

Descrição

A propriedade DropDownCount é declarada como uma variável inteira que define o número máximo de elementos a serem exibidos simultaneamente numa lista DropDown.

Exemplo

Você pode alterar a propriedade DropDownCount de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
ComboBox1.DropDownCount := 6;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TComboBox, TDBComboBox e TDBLookupCombo
```

Durante a execução do aplicativo:

```
TComboBox, TDBComboBox e TDBLookupCombo
```

DropDownWidth

Descrição

A propriedade DropDownWidth é declarada como uma variável inteira que define, em pixels, a largura da lista drop-down exibida pelo componente.

Exemplo

Você pode alterar a propriedade DropDownWidth diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
DBLookupComboBox1.DropDownWidth:= 6;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBLookupCombo e TDBLookupComboBox
```

Durante a execução do aplicativo:

```
TDBLookupCombo e TDBLookupComboBox
```

DropTarget

Descrição

Para componentes do tipo TListView, essa propriedade é declarada como uma variável do tipo TListItem e retorna o item sobre o qual outro está sendo solto em uma operação de "drag-drop". Essa propriedade só está definida durante a execução do aplicativo.

Para componentes do tipo TTreeView, essa propriedade é declarada como uma variável do tipo TTreeNode e retorna o item sobre o qual outro está sendo solto em uma operação de "drag-drop". Essa propriedade só está definida durante a execução do aplicativo.

Para componentes do tipo TTreeNode, essa propriedade é declarada como uma variável do tipo booleana, e especifica se o item está sendo o alvo em uma operação de "drag-drop". Essa propriedade só está definida durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TListView e TTreeView
```

EditFormat

Descrição

A propriedade EditFormat é declarada como uma variável do tipo string, usada para formatar o valor de um campo para fins de edição. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TIntegerField, TSmallintField e TWordField
```

EditMask

Descrição

A propriedade EditMask é uma variável do tipo string que define a máscara que limita os dados que podem ser digitados num componente do tipo EditMask ou em um campo de um registro de um banco de dados. A máscara limita os caracteres considerados válidos a serem digitados pelo usuário.

Uma máscara consiste em três campos separados por ponto-e-vírgula. O primeiro campo é a máscara propriamente dita; o segundo campo consiste num caractere que define se os caracteres ou a máscara devem ser salvos como parte dos dados; o terceiro campo define o caractere usado para representar espaços em branco na máscara.

Existe um conjunto de caracteres especiais para definir uma máscara:

- !: Se um ! aparecer na máscara, caracteres em branco não são armazenados como dados.
- >: Se um > aparecer na máscara, todos os caracteres seguintes estarão em letras maiúsculas, até que seja encontrado um caractere igual a <.
- <: Se um < aparecer na máscara, todos os caracteres seguintes estarão em letras minúsculas, até que seja encontrado um caractere igual a >.
- <>: Esse sinal significa que as letras maiúsculas e minúsculas serão armazenadas como digitadas pelo usuário.
- \: Os caracteres que seguem esse sinal serão tratados como caracteres literais.
- L: Esse caractere faz com que apenas caracteres alfabéticos (a-z e A-Z) sejam aceitos nessa posição.
- l: Esse caractere faz com que apenas caracteres alfabéticos (a-z e A-Z) sejam aceitos nessa posição, mas aceita que sejam omitidos.
- A: Esse caractere faz com que apenas caracteres alfanuméricos (0-9, a-z e A-Z) sejam aceitos nessa posição.
- a: Esse caractere faz com que apenas caracteres alfanuméricos (0-9, a-z e A-Z) sejam aceitos nessa posição, mas aceita que sejam omitidos.
- C: Esse caractere faz com que quaisquer caracteres sejam aceitos nessa posição.

- c: Esse caractere faz com que quaisquer caracteres sejam aceitos nessa posição.
- 0: Esse caractere faz com que apenas caracteres numéricos (0-9) sejam aceitos nessa posição.
- 9: Esse caractere faz com que apenas caracteres numéricos (0-9) sejam aceitos nessa posição, mas aceita que sejam omitidos.
- #: Esse caractere faz com que apenas caracteres numéricos (0-9) e sinais de mais (+) ou menos (-) sejam aceitos nessa posição, mas aceita que sejam omitidos.
- : Esse caractere (dois pontos) é usado para separar horas, minutos e segundos em dados horários. Se outro caractere separador de horas, minutos e segundos é definido nos atributos internacionais do Painel de Controle do seu sistema, esse caractere será usado em vez de dois pontos.
- /: Esse caractere é usado para separar meses, dias, horas, minutos e segundos em dados de datas. Se outro caractere separador de meses, dias e anos é definido nos atributos internacionais do Painel de Controle do seu sistema, ele será usado em vez do caractere /.
- ;: O ponto-e-vírgula é usado para separar máscaras.
- _: O caractere _ insere espaços em branco na caixa de edição.

Exemplo

Você pode alterar a propriedade EditMask diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
MaskEdit1.EditMask:= '999-999,1,@';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDateField, TDateTimeField, TmaskEdit, TStringField e TTimeField
```

Durante a execução do aplicativo:

```
TDateField, TDateTimeField, TmaskEdit, TStringField e TTimeField
```

EditMaskPtr

Descrição

A propriedade EditMaskPtr é declarada como uma variável do tipo string, usada como um ponteiro para a propriedade EditMask. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Na fase de projeto:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

EditMode

Descrição

A propriedade EditMode é uma variável booleana que define se a grade está ou não no modo de edição. Quando o seu valor é True, o usuário pode digitar texto em uma célula sem ter que antes pressionar Enter ou F2. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Durante a execução do aplicativo, você pode alterar o valor da propriedade EditMode com uma linha de código semelhante a:

```
DBCtrlGrid1.EditMode:= True;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBCtrlGrid
```

EditorMode

Descrição

Para componentes dos tipos TDBCtrlGrid, essa propriedade é declarada como uma variável booleana, e indica se o registro corrente do banco de dados está sendo editado pelo usuário.

Para componentes do tipo TDBGrid, TDrawGrid, TStringGrid e TDBLookupList, essa propriedade é declarada como uma variável booleana, e indica se o componente está no modo automático de edição. O valor dessa propriedade depende dos valores das propriedades Options e EditMode dos componentes.

Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBGrid, TDrawGrid, TStringGrid e TDBLookupList
```

EditText

Descrição

A propriedade EditText é uma variável do tipo string que armazena o valor da propriedade Text como ela é exibida durante a execução do aplicativo, com a máscara definida na propriedade EditMask. Em outras palavras, essa propriedade armazena o texto que é realmente exibido durante a execução do aplicativo. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que um componente Label1 do tipo TLabel exiba o valor da propriedade EditText de um componente DBEdit1 do tipo TDBEdit:

```
Label1.Caption := DBEdit1.EditText;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBEdit e TMaskEdit
```

Empty

Descrição

A propriedade Empty é declarada como uma variável booleana e determina se o objeto contém ou não um gráfico. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Você pode verificar o valor dessa propriedade durante a execução do aplicativo, incluindo uma linha de código como:

```
if Graphic1.Empty then ShowMessage('objeto vazio');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBitmap, TGraphic, Ticon e TMetafile
```

Enabled

Descrição

A propriedade Enabled é declarada como uma variável do tipo booleana e define se o componente responde a eventos do mouse, do teclado e do temporizador, isto é, define se o componente está habilitado.

Pata objetos da classe TAction, define o valor atribuído à propriedade Enabled dos controles e itens de menu associados ao objeto.

Exemplo

Se você quiser que um botão chamado Button1 só responda a um único clique do mouse, coloque a seguinte linha de código em seu evento OnClick:

```
Button1.Enabled := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles e os componentes, além de TAction, TMenuItem e TTimer.

Durante a execução do aplicativo:

Todos os controles e os componentes, além de TAction, TMenuItem e TTimer.

EnabledButtons

Descrição

A propriedade EnabledButtons é declarada como uma variável do tipo TButtonSet e consiste num conjunto que define os botões do componente que estão habilitados.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de um trecho de código.

O trecho de código a seguir faz com que todos os botões estejam habilitados:

```
TMediaPlayer1.EnabledButtons := [btPlay, btPause, btStop, btNext, btPrev, btStep, btBack,
btRecord, btEject]
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMediaPlayer
```

Durante a execução do aplicativo:

```
TMediaPlayer
```

EnableOpenBtn

Descrição

Essa propriedade é declarada como uma variável booleana e define se o botão Open file será exibido num formulário padrão de pré-visualização de relatórios.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TQRPrinter
```

EnablePrintBtn

Descrição

Essa propriedade é declarada como uma variável booleana e define se o botão Print será exibido num formulário padrão de pré-visualização de relatórios.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TQRPrinter
```

EnableSaveBtn

Descrição

Essa propriedade é declarada como uma variável booleana e define se o botão Save file será exibido num formulário padrão de pré-visualização de relatórios.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TQRPrinter
```

EndMargin

Descrição

A propriedade EndMargin é declarada como uma variável inteira e determina a distância, em pixels, da guia mais à direita dentre as visíveis no controle e a extremidade direita do controle.

Exemplo

O valor da propriedade EndMargin pode ser alterado diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
TabSet1.EndMargin := 6;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTabSet
```

Durante a execução do aplicativo:

```
TTabSet
```

EndPos

Descrição

A propriedade EndPos é declarada como uma variável do tipo inteiro longo (Longint) que define a posição final da execução ou gravação em um dispositivo multimídia. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que um arquivo de áudio do tipo WAV seja executado pela metade quando o usuário selecionar um botão chamado Button1 do tipo TButton:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     with MediaPlayer1 do
     begin
          FileName := 'D:\WINAPPS\SOUNDS\CARTOON.WAV';
          Open;
          EndPos := TrackLength[1] div 2;
          Play;
     end;
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

EOF

Descrição

A propriedade EOF é uma variável booleana que determina se um banco de dados está no seu último registro. Isto acontece depois que uma tabela é aberta e ocorre uma chamada aos eventos Last ou Next. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataset, TDecisionQuery, TIBDataset,
TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery, TDecisionQuery e TStoredProc
```

Error

Descrição

A propriedade Error é declarada como uma variável do tipo inteiro longo (Longint) que define o código do erro ocorrido na execução ou gravação em um dispositivo multimídia.

Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir exibe uma mensagem com o código do erro ocorrido durante a execução de um dispositivo multimídia.

```
ShowMessage('Erro '+IntToStr(MediaPlayer1.Error));
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

ErrorCode

Descrição

Para objetos das classes EDBClient e EReconcileError, a propriedade ErrorCode é declarada como uma variável do tipo string, e retorna o código de erro fornecido pelo BDE.

Para objetos das classes EoleException e EOleSysError, a propriedade ErrorCode é declarada como uma variável do tipo string, e retorna o código de erro numa operação OLE.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
EDBClient, EReconcileError , EOLEException, EOleSysError, EUpdateError
```

ErrorCount

Descrição

A propriedade ErrorCount é declarada como uma variável do tipo integer, e retorna o número de erros referenciados pela propriedade Errors.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
EDBEngineError
```

ErrorMessage

Descrição

A propriedade ErrorMessage é declarada como uma variável do tipo string que define a mensagem de erro correspondente ao código do erro ocorrido na execução ou gravação em um dispositivo multimídia. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir exibe uma mensagem com o código do erro ocorrido durante a execução de um dispositivo multimídia.

```
ShowMessage(MediaPlayer1.ErrorMessage);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

Errors

Descrição

A propriedade Errors é declarada como um array de objetos da classe TDBError e retorna os erros ocorridos com o BDE.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
EDBEngineError
```

Events

Descrição

A propriedade Events é declarada como uma lista de strings que armazena os nomes dos eventos interceptados pelo componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBEvents
```

Durante a execução do aplicativo:

```
TIBEvents
```

Exclusive

Descrição

A propriedade Exclusive é declarada como uma variável booleana e define se outros usuários podem ter acesso à mesma tabela que a representada por esse componente (isto não será possível se a propriedade tiver o valor True).

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
Table1.Exclusive := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTable
```

Durante a execução do aplicativo:

```
TTable
```

ExeName

Descrição

A propriedade ExeName é declarada como uma variável do tipo string que armazena o nome do arquivo executável da aplicação, incluindo-se o Path. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir exibe uma mensagem com o nome do arquivo executável da aplicação.

```
ShowMessage(Application.ExeName);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TApplication
```

Exists

Descrição

A propriedade Exists é declarada como uma variável booleana e define se a tabela referenciada pelo componente já existe ou se deve ser criada durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBTable, TTable
```

Expanded

Descrição

A propriedade Expanded é declarada como uma variável booleana e define se o item está expandido (se seus subitens estão sendo exibidos). Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TOutlineNode e TTreeNode
```

ExtendSelect

Descrição

A propriedade ExtendSelect é declarada como uma variável do tipo booleana e define se o usuário pode selecionar uma faixa de itens usando a tecla Shift, ou vários itens descontíguos usando a tecla Ctrl, desde que a propriedade Multiselect seja igual a True.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante uma linha de código, como:

```
ListBox1.ExtendSelect := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TListBox
```

Durante a execução do aplicativo:

```
TListBox
```

FieldClass

Descrição

A propriedade FieldClass é declarada como uma variável do tipo TFieldClass que define o tipo de componente correspondente ao campo. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TField
```

FieldCount

Descrição

A propriedade FieldCount é uma variável inteira que especifica o número de campos (controles) em um banco de dados. Esse número não corresponde necessariamente ao número de campos em uma tabela relacionada a esse banco de dados, pois podemos adicionar novos campos a uma tabela (cujos valores são calculados a partir dos existentes no banco de dados) ou remover campos com o Fields Designer. Em componentes do tipo TDBGrid e TDBLookupList, o valor dessa propriedade corresponde ao número de campos exibido pelo controle. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel mostre o número de colunas exibidas por um componente DBGrid1 do tipo TDBGrid.

```
Label1.Caption := IntToStr(DBGrid1.FieldCount);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TClientDataset, TDBGrid, TDBLookupList, TQuery, TDecisionQuery, TStoredProc, TTable
```

FieldDefs

Descrição

A propriedade FieldDefs é declarada como uma variável do tipo TFieldDef que armazena informações sobre cada campo em um banco de dados. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataset, TDecisionQuery, TIBDataset,
TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery, TDecisionQuery e TStoredProc
```

FieldList

Descrição

A propriedade FieldList é declarada como um objeto da classe TFieldList, que armazena uma lista de objetos descendentes de TField associados ao dataset.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataset, TDecisionQuery, TIBDataset,
TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery, TDecisionQuery e TStoredProc
```

FieldName

Descrição

A propriedade FieldName é declarada como uma variável do tipo string e define o nome da coluna que armazena o valor de um campo num banco de dados. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

FieldNo

Descrição

A propriedade FieldNo é declarada como uma variável inteira e define a posição ordinal de um campo num banco de dados. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFieldDef, TFloatField, TGraphicField, TIntegerField, TMemoField,
TSmallintField, TStringField, TTimeField, TVarBytesField e TWordField
```

Fields

Descrição

Para objetos do tipo TIndexDef, a propriedade Fields é declarada como uma variável do tipo string compreendendo os nomes e números que compõem o índice, separados por ponto-e-virgula (";").

Para objetos dos tipos TDBGrid, TDBLookupList, TQuery, TDecisionQuery, TStoredProc e TTable, a propriedade Fields é declarada como uma array do tipo TField e se refere a um campo específico de um banco de dados. O primeiro campo do banco de dados corresponde ao índice 0 da array.

Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que o texto mostrado no primeiro campo de um banco de dados seja exibido justificado à esquerda em um componente DBGrid1 do tipo

```
TBGrid.
DBGrid1.Fields[0].Alignment := taLeftJustify;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataset, TDBGrid, TDBLookupList,
TDecisionQuery, TIBDataset, TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TIndexDef,
TTable, TQuery, TDecisionQuery e TStoredProc
```

FieldValues

Descrição

A propriedade FieldValues é declarada como uma array que dá acesso direto aos valores armazenados em todos os campos armazenados na tabela representada pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TClientDataset
```

FILEEDIT

Descrição

A propriedade FileEdit é declarada como uma variável do tipo TEdit que define uma caixa de edição que exibirá o arquivo atualmente selecionado. Caso nenhum arquivo esteja selecionado, exibe a propriedade Mask.

Exemplo

Coloque um componente chamado FileListBox1 do tipo TFileListBox e um componente chamado Edit1 do tipo TDEdit em um formulário. Defina a propriedade FileEdit de FileListBox1 como Edit1 diretamente no Object Inspector ou mediante uma linha de código como:

```
FileListBox1.FileEdit := Edit1;
Isto faz com que Edit1 exiba sempre o arquivo selecionado em FileListBox1.
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TFileListBox
```

Durante a execução do aplicativo:

```
TFileListBox
```

FILEEDITSTYLE

Descrição

Essa propriedade define se deve ser oferecida ao usuário uma caixa de edição ou uma caixa combo para que o usuário digite o nome do arquivo.

Tabela de Valores:

Valor	Significado
fsEdit	Caixa de Edição.
fsComboBox	Caixa Combo.

Exemplo

Você pode alterar a propriedade FileEditStyle de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
OpenDialog1.FileEditStyle := fsEdit;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog
```

Durante a execução do aplicativo:

```
TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog
```

FILEEXTENSION

Descrição

A propriedade FileExtension é declarada como uma variável do tipo string que define a(s) extensão(ões) de arquivo default (separados por ponto-e-vírgula) para o relatório gerado pelo componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TRenderHTML, TRenderPDF, TRenderRTF e TRenderText
```

Durante a execução do aplicativo:

```
TRenderHTML, TRenderPDF, TRenderRTF e TRenderText
```

FileList

Descrição

A propriedade FileList é declarada como uma variável do tipo TFileListBox cuja função depende do componente ao qual se aplica.

Para componentes do tipo TDirectoryListBox, ela serve para fazer a conexão com um componente do tipo TFileListBox. Ao fazer esse tipo de conexão, o componente TFileListBox apontado por FileList exibirá sempre a lista de arquivos do diretório corrente.

Para componentes do tipo TFilterComboBox, ela serve para fazer a conexão com um componente do tipo TFileListBox. Ao fazer esse tipo de conexão, o componente TFileListBox exibe automaticamente os arquivos com a extensão definida no componente do tipo TFilterComboBox.

Exemplo

Coloque um componente chamado FileListBox1 do tipo TFileListBox e um componente chamado DirectoryListBox1 do tipo TDirectoryListBox em um Formulário. Defina a propriedade FileList de DirectoryListBox1 como FileListBox1 diretamente no Object Inspector ou mediante uma linha de código, como:

```
DirectoryListBox1.FileList := FileListBox1;
```

Isto faz com que FileListBox1 exiba sempre os arquivos do diretório apontado por

```
DirectoryListBox1.
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDirectoryListBox e TfilterComboBox
```

Durante a execução do aplicativo:

```
TDirectoryListBox e TFilterComboBox
```

FileName

Descrição

A propriedade FileName é declarada como uma variável do tipo string que define o arquivo selecionado no componente. Essa propriedade só está disponível durante a execução do aplicativo.

Para componentes do tipo TMediaPlayer, ela define o nome do arquivo a ser aberto pelo método Open ou salvo pelo comando Save.

Para componentes dos tipos TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog, essa propriedade é declarada como uma variável do tipo TFileName e define o nome do arquivo a ser exibido quando a aplicação exibe a caixa de diálogo.

Quando o usuário digita o nome de um arquivo e seleciona o botão OK, o nome do arquivo digitado é armazenado nessa propriedade.

Para objetos do tipo TIniFile, define o nome do arquivo INI sobre o qual serão realizadas operações de leitura e gravação.

Para componentes do tipo TAnimate, ela define o nome do arquivo que armazena o clipe de vídeo a ser exibido pelo componente.

Para componentes do tipo TClientDataset, define o nome do arquivo onde serão armazenadas as informações e os dados dos registros da tabela.

Para componentes dos tipos TBaseReport, TCanvasReport, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter, define o nome do arquivo que será criado durante a execução do relatório.

Componentes aos quais se aplica:

Na fase de projeto:

```
TAnimate, TClientDataset, TMediaPlayer, TOpenDialog, TOpenPictureDialog, TRvNDRWriter,
TRvRenderPreview, TRvRenderPrinter, TSaveDialog, TSavePictureDialog e TIniFile
```

Durante a execução do aplicativo:

```
TAnimate, TBaseReport, TCanvasReport, TClientDataset, TMediaPlayer, TOpenDialog,
TOpenPictureDialog, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter, TSaveDialog,
TSavePictureDialog e TIniFile
```

FILETYPE

Descrição

A propriedade FileType é declarada como uma variável do tipo TFileType que define os tipos de arquivo a serem exibidos pelo componente. Essa propriedade é, na realidade, composta por um conjunto de subpropriedades booleanas, listadas na tabela abaixo:

Tabela de Valores:

Valor	Significado
ftReadOnly	Se igual a True, exibe também os arquivos apenas de leitura.
ftHidden	Se igual a True, exibe também os arquivos ocultos.
ftSystem	Se igual a True, exibe também os arquivos de sistema.
ftVolumeID	Se igual a True, exibe também o nome do drive.
ftDirectory	Se igual a True, exibe também os diretórios.
ftArchive	Se igual a True, exibe também os arquivos comuns de trabalho.
ftNormal	Se igual a True, exibe qualquer tipo de arquivo.

Exemplo

Você pode alterar a propriedade FileType diretamente no Object Inspector ou mediante uma linha de código, como:

```
FileListBox1.FileType := ftNormal;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TFileListBox
```

Durante a execução do aplicativo:

```
TFileListBox
```

Filter

Descrição

Para componentes dos tipos TFilterComboBox, TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog, a propriedade Filter é declarada como uma variável do tipo string e define as máscaras de filtro de arquivo a serem exibidas pelo componente.

Para componentes dos tipos TClientDataSet, TIBTable e TTable, a propriedade Filter é declarada como uma variável do tipo string e define um filtro a ser aplicado à tabela representada pelo componente.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante uma linha de código, como:

```
FilterComboBox1.Filter := 'Text files|*.TXT';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TClientDataset, TTable, TFilterComboBox, TIBTable, TOpenDialog, TOpenPictureDialog e
TSaveDialog, TSavePictureDialog
```

Durante a execução do aplicativo:

```
TClientDataset, TTable, TFilterComboBox, TIBTable, TOpenDialog, TOpenPictureDialog e
TSaveDialog, TSavePictureDialog
```

Filtered

Descrição

Essa variável é declarada como uma variável booleana que define se há um filtro aplicado à tabela representada pelo componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TClientDataSet, TIBTable e TTable
```

Durante a execução do aplicativo:

```
TClientDataSet e TTable
```

FilterIndex

Descrição

A propriedade FilterIndex é declarada como uma variável inteira e define o filtro default (dentre os estabelecidos na propriedade Filter) a ser exibido na lista drop-down que define os tipos de arquivos selecionáveis.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante uma linha de código, como:

```
OpenDialog1.FilterIndex := 1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog
```

Durante a execução do aplicativo:

```
TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog
```

FindText

Descrição

A propriedade FindText é declarada como uma variável do tipo string e contém o texto a ser pesquisado pela aplicação.

Exemplo

Você pode definir o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
FindDialog1.FindText := 'Texto a ser pesquisado';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TFindDialog e TReplaceDialog
```

Durante a execução do aplicativo:

```
TFindDialog e TReplaceDialog
```

FirstIndex

Descrição

A propriedade FirstIndex é declarada como uma variável inteira que define a guia que vai aparecer mais à esquerda entre as guias visíveis no controle. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar a propriedade FirstIndex mediante a inclusão de uma linha de código, como:

```
TabSet1. FirstIndex = 1
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TTabSet
```

FirstPage

Descrição

A propriedade FirstPage é declarada como uma variável inteira que define o número da primeira página do intervalo de páginas a serem impressas de um relatório.

Componentes aos quais se aplica:

Na fase de Projeto:

```
TRvNDRWriter
```

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

FixedColor

Descrição

A propriedade FixedColor é declarada como uma variável do tipo TColor e especifica a cor das colunas e linhas fixas em um componente de grade.

Exemplo

Você pode alterar a propriedade FixedColor diretamente no Object Inspector ou mediante uma linha de código, como:

```
DrawGrid1.FixedColor = clBlue
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBGrid, TDrawGrid e TStringGrid
```

Durante a execução do aplicativo:

```
TDBGrid, TDrawGrid e TStringGrid
```

FixedCols

Descrição

A propriedade FixedCols é declarada como uma variável inteira que define o número de colunas fixas em um componente de grade.

Exemplo

Você pode alterar a propriedade FixedCols diretamente no Object Inspector ou mediante uma linha de código, como:

```
DrawGrid1.FixedCols = 1
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDrawGrid e TStringGrid
```

Durante a execução do aplicativo:

```
TDrawGrid TDecisionGrid e TStringGrid
```

FixedRows

Descrição

A propriedade FixedRows é declarada como uma variável inteira que define o número de linhas fixas em um componente de grade.

Exemplo

Você pode alterar a propriedade FixedRows diretamente no Object Inspector ou mediante uma linha de código, como:

```
DrawGrid1.FixedRows = 1
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDrawGrid TDecisionGrid e TstringGrid
```

Durante a execução do aplicativo:

```
TDrawGrid e TStringGrid
```

FOCUSCONTROL

Descrição

A propriedade FocusControl é declarada como uma variável do tipo TWinControl que define o controle que receberá o foco quando o usuário selecionar a combinação de teclas aceleradoras do componente (se existir).

Exemplo

Você pode alterar a propriedade FocusControl diretamente no Object Inspector ou mediante uma linha de código, como:

```
Label1.FocusControl = Edit1
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TLabel
```

Durante a execução do aplicativo:

```
TLabel
```

FOCUSED

Descrição

Essa variável é declarada como uma variável booleana que define se o componente possui ou não o foco. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TTreeNode e TListItem
```

FONT

Descrição

A propriedade Font é declarada como uma variável do tipo TFont e controla os atributos do texto exibido por um componente ou enviado para uma impressora. A propriedade Font possui as seguintes subpropriedades: Color, Name, Size e Style.

Exemplo

A linha de código seguinte altera a cor do texto exibido numa caixa de edição chamada

```
Edit1 para amarelo:
Edit1.Font.Color:= clYellow;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TCanvas, TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit,
TDBCtrlGrid, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo,
TDBRadioGroup, TDBText, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox,
TFilterComboBox, TFontDialog, TFrame, TForm, TGroupBox, THeader, TIWButton, TIWCheckBox,
TIWComboBox, TIWDBCheckBox, TIWDBComboBox, TIWDBEdit, TIWDBLookupComboBox, TIWDBLookupListBox,
TIWForm, TIWLabel, TIWListBox, TIWMemo, TLabel, TListBox, TMaskEdit, TMemo, TNotebook,
TOutline, TPaintBox, TPanel, TRadioButton, TScrollBox, TSpeedButton, TStringGrid,
TTabbedNotebook, TDBLookupComboBox, TDBLookupListBox, TTabControl, THeaderControl, TListView,
TPageControl, TStatusBar, TTreeView, TRichEdit, TQRBand, TQRCustomControl, TQRDBCalc,
TQRDBText, TQRLabel, TQRMemo, TQRPreview, TQRSysData e TTabSet
```

Durante a execução do aplicativo:

```
TCanvas, TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit,
TDBCtrlGrid, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo,
TDBRadioGroup, TDBText, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox,
TFilterComboBox, TFontDialog, TFrame, TForm, TGroupBox, THeader, TIWButton, TIWCheckBox,
TIWComboBox, TIWDBCheckBox, TIWDBComboBox, TIWDBEdit, TIWDBLookupComboBox, TIWDBLookupListBox,
TIWForm, TIWLabel, TIWListBox, TIWMemo, TLabel, TListBox, TMaskEdit, TMemo, TNotebook,
TOutline, TPaintBox, TPanel, TRadioButton, TScrollBox, TSpeedButton, TStringGrid,
TTabbedNotebook, TDBLookupComboBox, TDBLookupListBox, TTabControl, THeaderControl, TListView,
TPageControl, TStatusBar, TTreeView, TRichEdit, TQRBand, TQRCustomControl, TQRDBCalc,
TQRDBText, TQRLabel, TQRMemo, TQRPreview, TQRSysData e TTabSet
```

FONTS

Descrição

A propriedade Fonts é declarada como uma variável do tipo Tstrings, que retorna uma lista de strings com as fontes suportadas pelo componente. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir copia em um componente FontList do tipo TListBox a lista de fontes disponíveis no componente Screen do tipo TScreen (a tela).

```
var
      FontIndex: Integer;
begin
      FontList.Clear;
      FontList.Sorted := True;
      FontList.Items := Screen.Fonts;
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
Tprinter e TScreen
```

FOOTERBAND

Descrição

A propriedade FooterBand é declarada como uma variável do tipo TQRBand e define o objeto do tipo TQRBand exibido pelo componente após todos os registros da tabela principal.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRDetailLink
```

Durante a execução do aplicativo:

```
TQRDetailLink
```

ForceNewPage

Descrição

Essa propriedade é declarada como uma variável do tipo booleana e especifica se haverá sempre uma quebra de página antes da impressão de um objeto do tipo TQRBand.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
QRBand1.ForceNewPage:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRBand
```

Durante a execução do aplicativo:

```
TQRBand
```

FormatChars

Descrição

A propriedade FormatChars é declarada como uma variável booleana que define se os caracteres devem ser filtrados quando um texto é transferido numa conversação DDE.

Exemplo

Você pode alterar o valor da propriedade FormatChars de um componente diretamente no Object Inspector ou através de uma linha de código, como:

```
DdeClientConv1. FormatChars := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDDEClientConv
```

Durante a execução do aplicativo:

```
TDDEClientConv
```

Formats

Descrição

A propriedade Formats é um array que armazena os formatos mantidos pelo Clipboard.

Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Objetos aos quais se aplica:

Durante a execução do aplicativo:

```
TClipBoard
```

FormCount

Descrição

A propriedade FormCount é declarada como uma variável inteira e define o número de formulários existentes na aplicação. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo Tlabel exiba o número de formulários presentes em uma aplicação.

```
Label1.Caption := IntToStr(Screen.FormCount);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TScreen
```

Forms

Descrição

A propriedade Forms é declarada como uma array de formulários onde cada item da array é um dos formulários da aplicação. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir adiciona o nome de todos os formulários da aplicação em um componente ListBox1 do tipo TListBox.

```
var
      I: Integer;
begin
      For I := 0 to Screen.FormCount-1 do
      ListBox1.Items.Add(Screen.Forms[I].Name);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TScreen
```

FormStyle

Descrição

A propriedade FormStyle é declarada como uma variável do tipo TFormStyle que determina o estilo do formulário.

Tabela de Valores:

Valor	Significado
fsNormal	Definição padrão de formulário.
fsMDIChild	O formulário será uma janela-filha de uma aplicação MDI.
fsMDIForm	O formulário será o formulário-pai de uma aplicação MDI.
fsStayOnTop	O formulário permanece sobre todos os outros formulários do projeto, exceto aqueles que também têm a propriedade FormStyle igual a fsStayOnTop.

Exemplo

Você pode alterar o valor desta propriedade diretamente no Object Inspector.

Observação: Você não deve tentar alterar essa propriedade durante a execução do aplicativo, pois pode originar verdadeiros desastres.

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TForm
```

Frame

Descrição

Essa propriedade é declarada como uma variável do tipo TPen e especifica o objeto desse tipo usado para desenhar a moldura do componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRBand
```

Durante a execução do aplicativo:

```
TQRBand
```

FrameCount

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro e define o número de quadros que formam o clipe de vídeo exibido pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAnimate
```

FrameHeight

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro e define a altura, em pixels, dos quadros que formam o clipe de vídeo exibido pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAnimate
```

FrameWidth

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro e define a largura, em pixels, dos quadros que formam o clipe de vídeo exibido pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAnimate
```

Frames

Descrição

A propriedade Frames é declarada como uma variável do tipo inteiro longo (Longint) que define o número de quadros a ser avançado ou retrocedido com os métodos Back e Step. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir exibe uma mensagem com o valor da propriedade Frames de um componente do tipo TMediaPlayer.

```
ShowMessage('Frames = '+IntToStr(MediaPlayer1.Frames));
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

FreeOnTerminated

Descrição

Essa variável é declarada como uma variável booleana que define se o objeto deve ser destruído quando a Thread é finalizada.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TThread
```

Frequency

Descrição

A propriedade Frequency é declarada como uma variável inteira e define a freqüência com que as marcas (TickMarks) são exibidas pelo controle.

Componentes aos quais se aplica:

Na fase de projeto:

```
TTrackBar
```

Durante a execução do aplicativo:

```
TTrackBar
```

FromPage

Descrição

A propriedade FromPage é uma variável inteira que define o número da primeira página a ser impressa.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
PrintDialog1.FromPage:= 1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPrintDialog
```

Durante a execução do aplicativo:

```
TPrintDialog e TQRPrinter
```

FULLPATH

Descrição

A propriedade Fullpath é declarada como uma variável do tipo string e define se o path completo de um item em um componente do tipo TOutline, começa no nível 1. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TOutlineNode
```

GLYPH

Descrição

A propriedade Glyph é uma variável do tipo TBitmap que define o bitmap que aparece em um botão do tipo TBitBtn ou TSpeedButton. Você pode fornecer até quatro imagens em um botão do tipo TBitBtn ou TSpeedButton com um único arquivo Bitmap. O Delphi exibirá uma dessas imagens em correspondência ao estado do botão.

Tabela de Valores:

Posição da Imagem no Bitmap	Estado do botão	Descrição
Primeira	Liberado	Essa imagem aparece quando o botão não está selecionado ou quando não existem outras imagens definidas no Bitmap.
Segunda	Desabilitado	Essa imagem aparece quando o botão não está habilitado e portanto não pode ser selecionado.
Terceira	Pressionado	Essa imagem aparece enquanto o botão esquerdo do mouse estiver sendo pressionado sobre o botão, desde que este esteja habilitado.
Quarta	Mantido	Essa imagem aparece enquanto o botão permanece selecionado e só se aplica ao controle do tipo TSpeedButton.

Exemplo

O trecho de código a seguir carrega um bitmap num componente do tipo TSpeedButton chamado SpeedButton1 durante a execução do aplicativo, onde USER.BMP é um arquivo de bitmap definido pelo usuário.

```
SpeedButton1.Glyph.LoadFromFile('USER.BMP');
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn e TSpeedButton
```

Durante a execução do aplicativo:

```
TBitBtn e TSpeedButton
```

Graphic

Descrição

Essa propriedade é declarada como uma variável do tipo TGraphic e define o gráfico armazenado no componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TPicture
```

GridHeight

Descrição

A propriedade GridHeight é declarada como uma variável inteira que define a altura da grade, em pixels. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe a altura, em pixels, de um componente StringGrid1 do tipo TStringGrid quando o usuário seleciona uma célula com o mouse.

```
procedure TForm1.StringGrid1Click(Sender: TObject);
begin
      Label1.Caption := IntToStr(StringGrid1.GridHeigth);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDrawGrid e TStringGrid
```

GridLineWidth

Descrição

A propriedade GridLineWidth é declarada como uma variável inteira que define a largura, em pixels, da linha que divide as células da grade.

Exemplo

Você pode alterar a propriedade GridLineWidth diretamente no Object Inspector ou mediante uma linha de código, como:

```
StringGrid1.GridLineWidth := 1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDrawGrid e TStringGrid
```

Durante a execução do aplicativo:

```
TDrawGrid e TStringGrid
```

GridWidth

Descrição

A propriedade GridWidth é declarada como uma variável inteira que define a largura da grade, em pixels. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe a largura, em pixels, de um componente StringGrid1 do tipo TStringGrid quando o usuário seleciona uma célula com o mouse.

```
procedure TForm1.StringGrid1Click(Sender: TObject);
begin
      Label1.Caption := IntToStr(StringGrid1.GridWidth);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDrawGrid e TStringGrid
```

GroupIndex

Descrição

Para componentes do tipo TSpeedButton, a propriedade GroupIndex é declarada como uma variável inteira que define o grupo a que pertence um componente.

Para componentes do tipo TMenuItem, a propriedade GroupIndex é declarada como uma variável do tipo Byte que define os menus a serem mesclados em uma aplicação com vários formulários.

Exemplo

Você pode alterar a propriedade GroupIndex de um componente diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
SpeedButton1.GroupIndex := 2;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TSpeedButton
```

Durante a execução do aplicativo:

```
TSpeedButton
```

Handle

Descrição

A propriedade Handle lhe fornece um handle do Windows, de forma que você possa acessar objetos da GDI e usar funções da API do Windows que um handle como parâmetro. O tipo de handle fornecido vai depender do objeto que será usado. No caso de objetos de banco de dados, a propriedade Handle permite que você acesse as funções da API do Borland Database Engine (BDE). Essa propriedade só pode ser acessada durante a execução do aplicativo e não pode ter o seu valor diretamente alterado

pelo usuário. No caso de objetos de banco de dados do Interbase acessados pelo Interbase Express, a propriedade Handle permite que você acesse as funções da API deste mecanismo de acesso.

Tabela de Valores:

Tipo de Objeto	Tipo de Handle
TBitmap	HBitmap
TBrush	HBrush
TCanvas	HDC
TFont	HFont
TIcon	HIcon
TMetafile	HMetafile
TPen	HPen
TBitBtn	HWND
TDBNavigator	HWND
TMediaPlayer	HWND
TButton	HWND
TDBRadioGroup	HWND
TMemo	HWND
TCheckBox	HWND
TDirectoryListBox	HWND
TNotebook	HWND
TComboBox	HWND
TDrawGrid	HWND
TDBCtrlGrid	HWND
TDBCtrlPanel	HWND
TOLEContainer	HWND
TDBCheckBox	HWND
TDriveComboBox	HWND
TOutline	HWND
TDBComboBox	HWND
TEdit	HWND
TPanel	HWND
TDBEdit	HWND
TFileListBox	HWND
TRadioButton	HWND
TDBGrid	HWND
TFilterComboBox	HWND
TRadioGroup	HWND

continua

continuação

Tipo de Objeto	Tipo de Handle
TDBImage	HWND
TForm	HWND
TScrollBar	HWND
TDatabase	HDBIDB
TDBListBox	HWND
TGroupBox	HWND
TScrollBox	HWND
TDBLookupCombo	HWND
TDBLookupComboBox	HWND
THeader	HWND
TIBDatabase	TISC_DB_Handle
TIBTransaction	TISC_TR_HANDLE
TStringGrid	HWND
TDBLookupList	HWND
TDBLookupListBox	HWND
THotKey	HWND
TListBox	HWND
TListView	HWND
TTabbedNotebook	HWND
TStatusBar	HWND
TTabControl	HWND
TTabSheet	HWND
TTrackBar	HWND
TTreeView	HWND
TUpDown	HWND
TProgressBar	HWND
TRichEdit	HWND
TPageControl	HWND
TDBMemo	HWND
TMaskEdit	HWND
TTabSet	HWND
TApplication	HWND
TFindDialog	HWND
TReplaceDialog	HWND
TMainMenu	HMENU
TMenuItem	HMENU
TPopupMenu	HMENU

continua

continuação

Tipo de Objeto	Tipo de Handle
TPrinter	HDC
TDataBase	HDBIDB
TSession	HDBISES
TTable	HDBICur
TQuery, TDecisionQuery	HDBICur
TStoredProc	HDBICur

Exemplo

Coloque as seguintes linhas de código no evento OnPaint de um formulário, e será desenhado um círculo no seu interior.

Crie um novo projeto com dois formulários chamados Form1 e Form2; no Formulário Form1, insira um botão chamado Button1 e, no seu evento OnClick, coloque a seguinte linha de código (não se esqueça de incluir Unit2 na cláusula uses de Form1):

```
ShowWindow(Form2.Handle, SW_SHOWNORMAL)
```

Ao clicar com o mouse sobre Button1, será exibido na tela o formulário Form2.

É evidente que seria muito mais simples usar o código Form2.Show, mas o objetivo deste exemplo foi mostrar a utilização da propriedade handle na chamada a uma função da API do Windows.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os componentes relacionados na tabela de valores.

HandleIsShared

Descrição

A propriedade HandleIsShared é declarada como uma variável booleana e define se um handle (ponteiro) para um banco de dados é ou não compartilhado.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDatabase, TIBDatabase, TIBTransaction
```

HasItems

Descrição

A propriedade HasItems é declarada como uma variável booleana e define se um item de um componente TOutline possui ou não subitens. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TOutlineNode
```

HasProvider

Descrição

A propriedade HasProvider é declarada como uma variável booleana, e informa se o objeto recebe dados através de uma interface IProvider.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TClientDataset
```

Head1Font

Descrição

A propriedade Head1Font é declarada como um objeto da classe TFont, e define a fonte do texto de uma página HTML delimitado pelas tags <H1> e </H1>.

Componentes aos quais se aplica:

Na fase de projeto:

```
THTML
```

Durante a execução do aplicativo:

```
THTML
```

Head2Font

Descrição

A propriedade Head1Font é declarada como um objeto da classe TFont, e define a fonte do texto de uma página HTML delimitado pelas tags <H2> e </H2>.

Componentes aos quais se aplica:

Na fase de projeto:

```
THTML
```

Durante a execução do aplicativo:

```
THTML
```

Head3Font

Descrição

A propriedade Head1Font é declarada como um objeto da classe TFont, e define a fonte do texto de uma página HTML delimitado pelas tags <H3> e </H3>.

Componentes aos quais se aplica:

Na fase de projeto:

```
THTML
```

Durante a execução do aplicativo:

```
THTML
```

Head4Font

Descrição

A propriedade Head1Font é declarada como um objeto da classe TFont, e define a fonte do texto de uma página HTML delimitado pelas tags <H4> e </H4>.

Componentes aos quais se aplica:

Na fase de projeto:

`THTML`

Durante a execução do aplicativo:

`THTML`

Head5Font

Descrição

A propriedade Head1Font é declarada como um objeto da classe TFont, e define a fonte do texto de uma página HTML delimitado pelas Tags <H5> e </H5>.

Componentes aos quais se aplica:

Na fase de projeto:

`THTML`

Durante a execução do aplicativo:

`THTML`

Head6Font

Descrição

A propriedade Head1Font é declarada como um objeto da classe TFont, e define a fonte do texto de uma página HTML delimitado pelas Tags <H6> e </H6>.

Componentes aos quais se aplica:

Na fase de projeto:

`THTML`

Durante a execução do aplicativo:

`THTML`

HeaderBand

Descrição

A propriedade HeaderBand é declarada como uma variável do tipo TQRBand e define o objeto do tipo TQRBand exibido pelo componente.

Componentes aos quais se aplica:

Na fase de projeto:

`TQRDetailLink`

Durante a execução do aplicativo:

`TQRDetailLink`

Height

Descrição

A propriedade Height é uma variável inteira que define a dimensão vertical, em pixels, de um controle ou componente gráfico.

Para componentes do tipo TScreen, retorna a altura da tela, em pixels.

Para objetos do tipo TFont, define a altura da fonte, em pixels.

Exemplo

Para alterar a dimensão vertical de um formulário chamado Form1 durante a execução de um aplicativo, basta incluir a seguinte linha de código no evento correspondente:

```
Form1.Height := valor;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
Todos os controles.
```

Durante a execução do aplicativo:

Todos os controles; TBitmap, TFont, TGraphic, TIcon, TMetafile, TPicture e TScreen.

HelpContext

Descrição

A propriedade HelpContext é uma variável do tipo THelpContext que fornece um número ao qual está associado um tópico de auxílio sensível ao contexto. Ao ser pressionada a tecla F1 quando um componente possuir o foco, será exibida uma tela de auxílio correspondente ao número definido na propriedade HelpContext.

Exemplo

Você pode definir o valor da propriedade HelpContext diretamente no Object Inspector.

O trecho de código a seguir associa um arquivo de auxílio ao aplicativo e define para a propriedade HelpContext de um componente Edit1 do tipo TEdit o valor 7.

```
procedure TForm1.FormActivate(Sender: Tobject);
begin
     Application.HelpFile := 'HELP.HLP';
     Edit1.HelpContext := 7;
end;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
Todos os controles visuais, TColorDialog, TFindDialog, TAction, TFontDialog, TMenuItem,
TPopupMenu, TOpenDialog, TOpenPictureDialog, TPrintDialog, TPrinterSetupDialog, TReplaceDialog
e TSaveDialog, TSavePictureDialog
```

Durante a execução do aplicativo:

```
EAbort, EAbstractError, EAccessViolation, EarrayError, EassertinFailed, EbitsError,
EbrokerException, EcacheError, EclassNotFound, EcommomCalendarError, EcomponentError,
EcontrolC, EconvertError, EDatabaseError, EdateTimeError, EDBClient, EDBEditError,
EDBEngineError, EdimensionMapError, EDimIndexError, EdivByZero, EDSWriter, Eexternal,
EexternalException, EFCreateError, EfilerError, EFOpenError, EHeapException, EInOutError,
```

```
EInterpreterError, EintError, EIntCastError, EIntOverFlow, EInvalidArgument, EInvalidCast,
EInvalidGraphic, EInvalidGraphicOperation, EinvalidGridOperation, EinvalidImage, EinvalidOp,
EinvalidOperation, EinvalidPointer, ElistError, ElowCapacityError, EmathError,
EMCIDeviceError, EmenuError, EmonthCalError, EnoResultError, EoleCtrlError, EOLEError,
EoleException, EOleSysError, EoutLineError, EoutOfMemory, EoutOfResources, Eoverflow,
EpackageError, EparserError, Eprinter, Eprivilege, EpropertyError, EpropReadOnly,
EpropWriteOnky, ErangeError, EradError, EreconcileError, EregistryError, EresNotFound,
EsocketConnectionError, EsocketError, EstackOverflow, EstreamError, EstringListError, Ethread,
EtreeViewError, EunderFlow, EunSupportedTypeError, EUpdateError, EVariantError, EWin32Error,
EWriteError, Exception, EzeroDivide. Todos os controles visuais,:TAction, TColorDialog,
TFindDialog, TFontDialog, TMenuItem, TPopupMenu, TOpenDialog, TOpenPictureDialog,
TPrintDialog, TPrinterSetupDialog, TReplaceDialog e TSaveDialog, TSavePictureDialog
```

HelpFile

Descrição

Para objetos das classes Tapplication e TForm, a propriedade HelpFile é declarada como uma variável do tipo string que armazena o nome do arquivo de Help On-line da aplicação. Essa propriedade só está disponível durante a execução do aplicativo.

Para objetos da classe EOLEException, a propriedade HelpFile é declarada como uma variável do tipo string que armazena o nome do arquivo de Help On-line no qual está armazenada a descrição do erro. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir exibe uma mensagem com o nome do arquivo de auxílio da aplicação.

```
ShowMessage(Application.HelpFile);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TApplication, EOLEException
```

HelpKeyWord

Descrição

Essa propriedade é declarada como uma variável do tipo string e define a string de contexto do arquivo de Help associada ao controle ou componente.

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles e componentes.

Durante a execução do aplicativo:

Todos os controles e componentes.

HelpType

Descrição

Essa propriedade é declarada como uma variável do tipo ThelpType, que define como controles e itens de menu acessarão o arquivio de Help vinculado à aplicação. Pode assumir o valor htKeyword (para identificar o tópico por uma string de contexto) ou htContent (para identificar o tópico pelo valor numérico a ele associado).

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles e componentes.

Durante a execução do aplicativo:

Todos os controles e componentes.

HideScrollBars

Descrição

Essa propriedade é declarada como uma variável do tipo booleana que define se as barras de rolagem não devem ser exibidas quando não forem necessárias. Seu valor default é igual a True.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou durante a execução do aplicativo, mediante a inclusão de uma linha de código como:

```
RichEdit1.HideScrollBars:= False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TRichEdit
```

Durante a execução do aplicativo:

```
TRichEdit
```

HideSelection

Descrição

A propriedade HideSelection é uma variável booleana que define se um texto selecionado permanece selecionado quando o componente perde o foco.

Exemplo

Você pode alterar a propriedade HideSelection diretamente no Object Inspector ou mediante uma linha de código, como:

```
Edit1.HideSelection := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TEdit e TMemo
```

Durante a execução do aplicativo:

```
TEdit e TMemo
```

Hint

Descrição

A propriedade Hint é uma variável string que pode ser usada para exibir um texto de auxílio quando o usuário mantém o ponteiro do mouse sobre um controle ou quando ocorre um evento OnHint.

No primeiro caso, basta definir o valor da propriedade ShowHint como True e o valor da propriedade Hint como a string a ser exibida no Object Inspector (durante a fase do projeto) ou mediante código (durante a execução do aplicativo).

No segundo caso, o código do evento OnHint do objeto TApplication será responsável pela exibição da string em um controle ou item de menu (Veja mais detalhes sobre o evento OnHint na seção Eventos deste guia).

Exemplo

Você pode alterar a propriedade Hint diretamente no Object Inspector ou mediante um trecho de código. Para alterar a propriedade Hint de um botão chamado Button1 de um formulário chamado Form1, inclua a seguinte linha de código:

```
Form1.Button1.Hint := 'String de Auxílio';
```

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles, além de objetos da classe TAction.

Durante a execução do aplicativo:

Todos os controles e os componentes, além de objetos da classe TAction, TApplication e TMenuItem.

HintColor

Descrição

A propriedade HintColor é declarada como uma variável do tipo TColor que define a cor dos boxes que circundam o texto de auxílio definido na propriedade Hint de um controle de uma aplicação. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir define como azul a cor dos boxes de auxílio dos controles de uma aplicação.

```
Application.HintColr := clBlue;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TApplication
```

HintPause

Descrição

A propriedade HintPause é declarada como uma variável inteira que define o intervalo de tempo (em milissegundos) que o ponteiro do mouse deve permanecer sobre um controle de uma aplicação para que seja exibida a string de auxílio definida na sua propriedade Hint. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir define como sendo igual a 1 segundo (1000 milissegundos) o valor da propriedade HintPause dos controles de uma aplicação.

```
Application.HintPause := 1000;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TApplication
```

Hints

Descrição

A propriedade Hints é declarada como uma variável do tipo Tstrings que associa, mediante uma lista de strings, um texto de auxílio a cada um dos botões do controle, a ser exibido quando o usuário mantém o ponteiro do mouse sobre um botão ou quando ocorre um evento OnHint.

Exemplo

Você pode definir essa propriedade diretamente no Object Inspector com o String List Editor. Nesse caso, a primeira linha será o texto de auxílio para o primeiro botão (o da esquerda), a segunda linha será o texto de auxílio para o segundo botão (o segundo, da esquerda para a direita) e assim por diante.

O trecho de código a seguir altera o texto de auxílio do primeiro botão de um controle DBNavigator1 do tipo TDBNavigator, quando o usuário dá um clique com o mouse sobre um formulário chamado Form1:

```
procedure TForm1.FormClick(Sender: TObject);
var
  NovaHints : TStringList;
begin
NovaHints := TStringList.Create;
  NovaHints.Add('Texto de auxílio do primeiro botão');
  DBNavigator1.Hints := NovaHints;
end;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBNavigator
```

Durante a execução do aplicativo:

```
TDBNavigator
```

HistoryList

Descrição

A propriedade HistoryList é declarada como uma variável do tipo Tstrings e consiste em uma lista de strings que define o nome dos arquivos a serem exibidos na lista drop-down da caixa combo File Name quando a caixa de diálogo é aberta, e desde que a propriedade FileEditStyle da caixa de diálogo seja fsComboBox.

Componentes aos quais se aplica:

Na fase de projeto:

```
TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog
```

Durante a execução do aplicativo:

```
TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog
```

HorizAxis

Descrição

A propriedade HorizAxis é declarada como uma variável do tipo ThorizAxis e define o eixo horizontal cuja escala será usada na exibição dos pontos da série.

O tipo ThorizAxis é definido da seguinte maneira:

```
THorizAxis = (aTopAxis,aBottomAxis);
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TChartSeries
```

Durante a execução do aplicativo:

```
TChartSeries
```

HorzScrollBar

Descrição

A propriedade HorzScrollBar é declarada como uma variável do tipo TControlScrollBar, cujas subpropriedades definem o comportamento de uma barra de rolagem horizontal em um componente dos tipos TForm, TFrame ou TScrollBox.

Exemplo

Você pode alterar as subpropriedades da propriedade HorzScrollBar de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
Form1.HorzScrollBar.Visible := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm, TFrame e TScrollBox
```

Durante a execução do aplicativo:

```
TForm, TFrame e TScrollBox
```

Host

Descrição

Essa propriedade é declarada como uma variável do tipo string e define o nome ou endereço IP de um servidor remoto.

Componentes aos quais se aplica:

Na fase de projeto:

```
TNMDayTime, TNMEcho, TNMFinger, TNMFTP, TNMHTTP, TNMMSG, TNMNNTP, TNMPOP3, TNMSMTP, TNMSTRM,
TNMTime, TPowerSock
```

Durante a execução do aplicativo:

```
TNMDayTime, TNMEcho, TNMFinger, TNMFTP, TNMHTTP, TNMMSG, TNMNNTP, TNMPOP3, TNMSMTP, TNMSTRM,
TNMTime, TPowerSock
```

HotKey

Descrição

A propriedade HotKey é declarada como uma variável do tipo TShortCut e define a combinação de teclas usada como tecla aceleradora.

Componentes aos quais se aplica:

Na fase de projeto:

```
THotKey
```

Durante a execução do aplicativo:

```
THotKey
```

Icon

Descrição

A propriedade Icon é declarada como uma variável do tipo TIcon que define o ícone a ser exibido quando a aplicação for minimizada. Se nenhum valor for atribuído a essa propriedade, será usado o ícone padrão da aplicação. Para objetos do tipo TPicture, é declarada como uma variável do tipo TIcon e representa o objeto do tipo TIcon armazenado.

Exemplo

Você pode alterar a propriedade Icon de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo um trecho de código como:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Icon.LoadFromFile('ICONE.ICO');
end;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TApplication e TForm
```

IdleTimer

Descrição

Esta propriedade é declarada como uma variável do tipo inteiro, e define o tempo que o banco de dados deve aguardar antes de terminar uma conexão ociosa.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDatabse, TIBTransaction
```

Durante a execução do aplicativo:

```
TIBDatabse, TIBTransaction
```

ImageIndex

Descrição

A propriedade ImageIndex é declarada como uma variável inteira e define a imagem de um componente ImageList a ser exibida nos controles e itens de menu associados ao objeto.

Componentes aos quais se aplica:

Na fase de projeto:

```
TAction, TMenuItem
```

Durante a execução do aplicativo:

```
TAction, TMenuItem
```

Images

Descrição

A propriedade Images é declarada como uma variável do tipo TImageList e define a lista de imagens (bitmaps) associada ao componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TAction, TMainMenu, TPopupMenu, TTreeView
```

Durante a execução do aplicativo:

```
TAction, TMainMenu, TPopupMenu, TTreeView
```

ImageType

Descrição

A propriedade ImageType é declarada como uma variável do tipo TImageType e define se o controle desenhará a imagem ou a máscara a ela associada.

Tabela de Valores:

Valor	Significado
itImage	Desenha a imagem.
itMask	Desenha a máscara.

Componentes aos quais se aplica:

Na fase de projeto:

```
TImageList
```

Durante a execução do aplicativo:

```
TImageList
```

Increment

Descrição

Para componentes do tipo TControlScrollBar, a propriedade Increment é uma variável inteira que define quantas posições a caixa de rolagem deve se mover em uma barra de rolagem quando o usuário seleciona uma das setas nas extremidades de uma barra de rolagem.

Para componentes do tipo TUpDown, essa propriedade é definida como uma variável booleana e determina a variação do valor armazenado na propriedade Position quando o usuário seleciona com o botão esquerdo do mouse uma das setas do controle.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
Form1.HorzScrollBar.Range:= 8;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TControlScrollBar e TUpDown
```

Durante a execução do aplicativo:

```
TControlScrollBar e TUpDown
```

INDENT

Descrição

A propriedade Indent é declarada como uma variável inteira e define a endentação, em pixels, a ser usada na exibição de subitens quando uma lista de itens é expandida.

Componentes aos quais se aplica:

Na fase de projeto:

```
TTreeView
```

Durante a execução do aplicativo:

```
TTreeView
```

INDEX

Descrição

A propriedade Index é declarada como uma variável inteira e define o índice de um campo na propriedade Fields de um banco de dados. Para componentes do tipo TOutlineNode, é declarada como uma variável do tipo inteiro longo (Longint) e define seu índice dentro do componente TOutline que o contém.

Para objetos da classe TAction, define o índice que o identifica entre as ações definidas para um componente TactionList.

Exemplo

Você pode alterar o valor dessa propriedade na fase de projeto com o Fields Editor.

Componentes aos quais se aplica:

Na fase de projeto:

```
TAction, TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField,
TDateField, TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField,
TSmallintField, TStringField, TTimeField, TVarBytesField e TWordField
```

Durante a execução do aplicativo:

```
TAction, TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField,
TDateField, TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField,
TSmallintField, TStringField, TTimeField, TVarBytesField, TOutlineNode e TWordField
```

INDEXDEFS

Descrição

A propriedade IndexDefs é declarada como uma variável do tipo TIndexDefs e armazena informações sobre todos os índices da tabela. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Na fase de projeto:

```
TClientDataSet
```

Durante a execução do aplicativo:

```
TIBTable, TTable, TClientDataSet
```

IndexFieldCount

Descrição

Essa propriedade é declarada como uma variável inteira que armazena o número de campos com o índice corrente. Ela só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBTable, TTable, TClientDataSet
```

IndexFieldNames

Descrição

Essa propriedade é declarada como uma variável do tipo string, usada com um servidor SQL, para identificar as colunas a serem usadas como índice na tabela.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOTable, TIBTable, TTable, TClientDataSet
```

Durante a execução do aplicativo:

```
TADOTable, TTable, TclientDataSet
```

IndexFields

Descrição

Essa propriedade é declarada como uma variável do tipo TField e dá acesso à informação armazenada em cada campo no índice corrente do banco de dados. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Na fase de projeto:

```
TClientDataSet
```

Durante a execução do aplicativo:

```
TIBTable, TTable, TClientDataSet
```

IndexName

Descrição

Essa propriedade é declarada como uma variável do tipo string, usada para definir um índice secundário da tabela.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBTable, TTable, TClientDataSet
```

Durante a execução do aplicativo:

```
TIBTable, TTable, TClientDataSet
```

INITIALDIR

Descrição

A propriedade InitialDir é declarada como uma variável do tipo string que define o diretório atual quando a caixa de diálogo é aberta.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
OpenDialog1.InitialDir := 'C:\delphi\teste';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog
```

Durante a execução do aplicativo:

```
TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog
```

INPLACEACTIVE

Descrição

A propriedade InPlaceActive é uma variável do tipo booleana que determina se um objeto OLE está ativo. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir exibe uma mensagem se InPlaceActive for igual a True:

```
if InPlaceActive = True then ShowMessage('Objeto Ativo');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TOLEContainer
```

INSERTSQL

Descrição

Essa propriedade armazena uma lista de strings contendo a declaração SQL usada para inserir um registro.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataset, TIBUpdateSQL, TUpdateSQL
```

Durante a execução do aplicativo:

```
TIBDataset, TIBUpdateSQL, TUpdateSQL
```

IntegralHeight

Descrição

A propriedade IntegralHeight é uma variável booleana que define como o componente será exibido no formulário. Se for igual a True, sua altura é ajustada de forma que nenhum dos itens seja exibido parcialmente. Se a propriedade Style do componente tiver o valor igual a lbOwerDrawVariable, a propriedade ItemHeight não tem nenhum efeito.

Exemplo

Você pode alterar a propriedade ItemHeight de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
ListBox1.IntegralHeight := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBListBox, TDirectoryListBox, TFileListBox e TListBox
```

Durante a execução do aplicativo:

```
TDBListBox, TDirectoryListBox, TFileListBox e TListBox
```

Interval

Descrição

A propriedade Interval é declarada como uma variável inteira e define o intervalo de tempo, em milissegundos, para ativação do evento OnTimer.

Exemplo

O valor da propriedade EndMargin pode ser alterado diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
Timer1.Interval := 6000;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTimer
```

Durante a execução do aplicativo:

```
TTimer
```

Intransaction

Descrição

A propriedade Intransaction é uma variável booleana que define se a transação representada pelo componente está em progresso.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBTransaction
```

Durante a execução do aplicativo:

```
TIBTransaction
```

INVALIDKEYS

Descrição

A propriedade InvalidKeys é declarada como uma variável do tipo THKInvalidKeys e define um conjunto de teclas consideradas inválidas:

Tabela de Valores:

Valor	Significado
hcNone	Teclas inalteradas são consideradas inválidas.
hcShift	A tecla Shift é considerada inválida.
hcCtrl	A tecla Ctrl é considerada inválida.
hcAlt	A tecla Alt é considerada inválida.
hcShiftCtrl	A combinação de teclas Shift+Ctrl é considerada inválida.
hcShiftAlt	A combinação de teclas Shift+Alt é considerada inválida.
hcCtrlAlt	A combinação de teclas Ctrl+Alt é considerada inválida.
hcShiftCtrlAlt	A combinação de teclas Shift+Ctrl+Alt é considerada inválida.

Componentes aos quais se aplica:

Na fase de projeto:

```
THotKey
```

Durante a execução do aplicativo:

```
THotKey
```

ISINDEXFIELD

Descrição

A propriedade IsIndexField é declarada como uma variável booleana, que define se um campo é ou não indexado. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

ISMASKED

Descrição

A propriedade IsMasked é uma variável booleana que define se o componente possui ou não uma máscara para exibição de texto.

Exemplo

Você pode alterar a propriedade IsMasked de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
MaskEdit1.EditMask := '!99/99/00;1;_';
```

O trecho de código acima define uma máscara para exibição de texto no formato

```
MM/DD/YY.
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBEdit e TMaskEdit
```

Durante a execução do aplicativo:

```
TDBEdit e TMaskEdit
```

IsNull

Descrição

A propriedade IsNull é declarada como uma variável booleana, que define se o valor de um campo é igual a Null. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir exibe uma mensagem se o valor armazenado em um campo GraphicField1 do tipo TGraphicField é igual a Null:

```
if GraphicField1,IsNull = Null then ShowMessage('O valor armazenado no campo é NULL');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

IsReadOnly

Descrição

Esta propriedade é declarada como uma variável do tipo inteiro, e define se o banco de dados deve ser acessado apenas para leitura (apenas para a versão 6 do Interbase).

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBDatabse
```

IsSQLBased

Descrição

A propriedade IsSQLBased é declarada como uma variável booleana, que define se o driver usado é diferente de STANDARD. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDatabase
```

IsVisible

Descrição

A propriedade IsVisible é declarada como uma variável booleana e define se um item está visível dentro do componente TOutline que o contém. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TOutlineNode
```

ItemCount

Descrição

A propriedade ItemCount é declarada como uma variável do tipo inteiro longo (Longint) que define o número de itens de um componente do tipo TOutline. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Você pode exibir o número de itens em um componente Outline1 do tipo TOutline em um componente Label1 do tipo TLabel com a seguinte linha de código:

```
Label1.Caption := IntToStr(Outline1.ItemCount);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TOutline
```

ItemHeight

Descrição

A propriedade ItemHeight é uma variável inteira que define a altura dos itens de componentes dos tipos TComboBox, TDBComboBox, TDBListBox, TDirectoryListBox, TFileListBox, TListBox e TOutline, quando os itens têm altura fixa, o que é definido pela propriedade Style de cada componente. A propriedade Style deve ser igual a:

- lsOwnerDrawFixed, para componentes do tipo TListBox, TDBListBox, TFileListBox e TDirectoryListBox;
- lsOwnerDrawFixed, para componentes do tipo TComboBox e TDBComboBox;
- lsOwnerDrawFixed, para componentes do tipo TOutline.

Exemplo

Você pode alterar a propriedade ItemHeight de um componente diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
ComboBox1.ItemHeight := 20;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TComboBox, TDBComboBox, TDBListBox, TDirectoryListBox, TFileListBox, TListBox e TOutline.
```

Durante a execução do aplicativo:

```
TComboBox, TDBComboBox, TDBListBox, TDirectoryListBox, TFileListBox, TListBox e TOutline
```

ItemIndex

Descrição

A propriedade ItemIndex é uma variável inteira que define a posição de um item selecionado dentro da lista de itens do componente. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode atribuir o número que identifica a posição do item selecionado em uma caixa de listagem chamada ListBox1 a uma variável inteira "a" durante a execução do aplicativo, incluindo a seguinte linha de código:

```
a := ListBox1.ItemIndex;
```

Se nenhum item estiver selecionado, ItemIndex retorna o valor -1.

O primeiro item da lista ocupa a posição 0, isto é, se uma lista tem n itens, então o n-ézimo item ocupará a posição n-1. Se a propriedade MultiSelect do componente for True e houver mais de um elemento selecionado, ItemIndex retornará a posição do elemento que possui o foco.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TComboBox, TDBComboBox, TDBRadioGroup, TDirectoryListBox, TDriveComboBox, TFileListBox,
TFilterComboBox, TListBox e TRadioGroup
```

Items

Descrição

Para componentes dos tipos TComboBox, TDBRadioGroup, TDirectoryListBox, TDriveComboBox, TFileListBox, TFilterComboBox, TListBox e TRadioGroup, essa propriedade é uma variável do tipo TStrings que define a lista de strings que aparecem no componente. A propriedade Items é por si só um objeto do tipo Tstrings, no qual você pode adicionar, remover, inserir e mover componentes com os métodos Add, Delete, Insert, Exchange e Move.

Para componentes do tipo TListView, essa propriedade é uma variável do tipo TListItems, que define a lista de itens que aparecem no componente.

Para componentes do tipo TTreeView, essa propriedade é uma variável do tipo TTreeNodes, que define a lista de itens que aparecem no componente.

Para objetos do tipo TList, essa propriedade é declarada como uma array de ponteiros que permite o acesso direto aos ponteiros da lista,

Exemplo

Você pode adicionar uma string 'novo item' a um componente ComboBox1 do tipo TComboBox com a seguinte linha de código:

```
ComboBox1.Items.Add('novo item');
```

Na fase de projeto, você pode operar sobre os itens do componente acionando o String

```
List Editor, na propriedade Items do Object Inspector.
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TComboBox, TDBCombobox, TDBRadioGroup, TDirectoryListBox, TDriveComboBox, TFileListBox,
TFilterComboBox, TIWComboBox, TIWDBCombobox, TIWList, TIWListBox, TListBox, TIWTreeview,
TListView, TTreeView e TRadioGroup
```

Durante a execução do aplicativo:

```
TComboBox, TDBCombobox, TDBRadioGroup, TDirectoryListBox, TDriveComboBox, TFileListBox,
TFilterComboBox, TIWComboBox, TIWDBCombobox, TIWList, TIWListBox, TListBox, TIWTreeview,
TListView, TTreeView e TRadioGroup
```

ITEMSEPARATOR

Descrição

A propriedade ItemSeparator é declarada como uma variável string que define a string usada para separar os itens em um componente do tipo TOutline.

Exemplo

Você pode alterar a propriedade ItemSeparator diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
Outline1.ItemSeparator := '\';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOutline
```

Durante a execução do aplicativo:

```
TOutline
```

KEEPCONNECTION

Descrição

Essa propriedade é declarada como uma variável Booleana e indica se a conexão com um componente do tipo TSQLConnection deve ser mantida quando não houver tabelas abertas.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TSQLConnection
```

KEEPCONNECTIONS

Descrição

Essa propriedade é declarada como uma variável Booleana e indica se a conexão com um componente do tipo TDataBase deve ser mantida quando não houver tabelas abertas. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar o valor dessa propriedade mediante a inclusão de uma linha de código como:

```
Session.KeepConnections := False;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TSession
```

KeyExclusive

Descrição

A propriedade KeyExclusive é uma variável booleana que define se funções de pesquisa e busca excluirão os registros especificados nas funções FindNearest, GotoNearest, EditRangeStart e EditRangeEnd. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TTable, TclientDataSet
```

KeyField

Descrição

A propriedade KeyField é declarada como uma variável do tipo string e define o nome do campo do banco de dados definido na sua propriedade ListSource que corresponde ao campo definido na propriedade DataField.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBLookupListBox e TDBLookupComboBox
```

Durante a execução do aplicativo:

```
TDBLookupListBox e TDBLookupComboBox
```

KeyFieldCount

Descrição

A propriedade KeyFieldCount é uma variável booleana que define o número de campos-chave usados pelas funções de busca (GotoKey, FindKey, EditKey, etc.).

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TTable e TClientDataset
```

KeyPreview

Descrição

A propriedade KeyPreview é uma variável booleana que define se o formulário deve ou não responder a um pressionamento de tecla, através dos eventos OnKeyDown, OnKeyPress e OnKeyUp, em vez de acionar os eventos do componente selecionado.

Exemplo

Você pode alterar a propriedade KeyPreview de um formulário diretamente no Object Inspector ou durante a execução do aplicativo, incluindo uma linha de código como:

```
Form1.KeyPreview := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TForm
```

KEYVALUE

Descrição

A propriedade KeyValue é declarada como uma variável do tipo Variant e reflete o valor armazenado no campo especificado na propriedade KeyField. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBLookupListBox e TDBLookupComboBox
```

KEYVIOLCOUNT

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro longo (Longint) que define o número de registros que deixaram de sofrer uma operação pelo método Execute de um componente do tipo TBatchMove, devido a ocorrência de um erro de violação de integridade. Essa propriedade só está definida durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir verifica se houve violação de integridade durante a execução do método Execute de um componente BatchMove1 do tipo TBatchMove:

```
BatchMove1.Execute;
if BatchMove1.KeyViolCount <> 0 then ShowMessage('ocorreram violações em '+
IntToStr(KeyViolCount)+ ' registros ');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBatchMove
```

KEYVIOLTABLENAME

Descrição

Essa propriedade é declarada como uma variável do tipo TFileName que cria uma tabela local, do tipo Paradox, com todos os registros que sofreram violação de integridade durante uma operação pelo método Execute de um componente do tipo TBatchMove.

Componentes aos quais se aplica:

Na fase de projeto:

```
TBatchMove
```

Durante a execução do aplicativo:

```
TBatchMove
```

Kind

Descrição

Para componentes do tipo TBitBtn, a propriedade Kind é uma variável do tipo TBitBtnKind que define o tipo de Bitmap exibido pelo botão.

Para componentes do tipo TScrollBar, a propriedade Kind é uma variável do tipo TScrollBarKind que define o tipo de barra de rolagem.

Tabela de Valores para componentes do Tipo TBitBtn:

Valor	Significado
bkCustom	Bitmap definido pelo usuário.
bkOK	Botão OK padrão, com uma marca de verificação na cor verde e propriedade Default igual a True.
bkCancel	Botão Cancel padrão, com um "x" na cor vermelha e propriedade Cancel igual a True.
bkYes	Botão Yes padrão, com uma marca de verificação na cor verde e propriedade Default igual a True. A diferença entre esse tipo de botão e o botão OK é que, nesse caso, quando o usuário clica sobre o botão, as alterações feitas na caixa de diálogo são aceitas antes que ela se feche.
bkNo	Botão No padrão, com uma marca vermelha representando um círculo cortado e propriedade Cancel igual a True. A diferença desse tipo de botão e o botão Cancel é que, nesse caso, quando o usuário clica sobre o botão, as alterações feitas na caixa de diálogo são descartadas antes que ela se feche.
bkHelp	Botão de auxílio padrão, com uma marca de interrogação na cor cyan. Quando o usuário clica sobre o botão, uma tela de auxílio *deve* ser exibida (eu disse *deve*, pois isso depende de trabalho de codificação feito pelo programador).
bkClose	Botão Close padrão, com o desenho de uma porta e um sinal de "Exit" e propriedade Default igual a True. Quando o usuário clica sobre o botão, o formulário a que ele pertence se fecha.
bkAbort	Botão Abort padrão, com um "x" na cor vermelha e propriedade Cancel igual a True.
bkRetry	Botão Retry padrão, com uma seta circular verde.
bkIgnore	Botão Ignore padrão, com o desenho de um homem verde se afastando.
bkAll	Botão All padrão, com uma marca de verificação dupla na cor verde e propriedade default igual a True.

Tabela de Valores para componentes do Tipo TScrollBar:

Valor	Significado
sbHorizontal	A barra de rolagem é horizontal.
sbVertical	A barra de rolagem é vertical.

Exemplo

A seguinte linha de código define como bkOk a propriedade Kind de um botão chamado BitBtn1 do tipo TBitBtn

```
BitBtn1.Kind := bkOK;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn e TScrollBar
```

Durante a execução do aplicativo:

```
TBitBtn e TScrollBar
```

LargeChange

Descrição

A propriedade LargeChange é declarada como uma variável do tipo TScrollBarInc que define quantas posições devem ser deslocadas na barra de rolagem quando o usuário dá um clique nela em qualquer um dos lados da caixa de rolagem ou pressiona as teclas PageUp e PageDn.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
ScrollBar1.LargeChange := 100
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TScrollBar
```

Durante a execução do aplicativo:

```
TScrollBar
```

LargeImages

Descrição

Essa propriedade define as imagens exibidas pelo componente quando a sua propriedade ViewStyle possui o valor vsIcon.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TListView
```

LastPage

Descrição

A propriedade LastPage é declarada como uma variável inteira que define o número da última página do intervalo de páginas a serem impressas de um relatório.

Componentes aos quais se aplica:

Na fase de Projeto:

```
TRvNDRWriter
```

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

Layout

Descrição

A propriedade Layout é uma variável do tipo TButtonLayout que define onde o bitmap aparece em um botão do tipo TBitBtn ou TSpeedButton.

Tabela de Valores:

Valor	Significado
blGlyphLeft	A imagem aparece próxima ao lado esquerdo do botão.
blGlyphRight	A imagem aparece próxima ao lado direito do botão.
blGlyphTop	A imagem aparece próxima ao topo do botão.
blGlyphBottom	A imagem aparece próxima à base do botão.

Exemplo

O trecho de código a seguir define a propriedade Layout de um botão chamado BitBtn1 do tipo TBitBtn como blGlyphLeft:

```
BitBtn1.Layout := blGlyphLeft;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn e TSpeedButton
```

Durante a execução do aplicativo:

```
TBitBtn e TSpeedButton
```

Left

Descrição

A propriedade Left é uma variável inteira que define, em pixels, a coordenada da extremidade esquerda de um componente em relação à extremidade esquerda do formulário que o contém. No caso de um formulário, essa propriedade é medida em relação à tela.

Exemplo

Para alterar a propriedade Left de um botão chamado Button1 durante a execução de um aplicativo, basta incluir a seguinte linha de código no evento correspondente:

```
Button1.Left:= valor;
```

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles.

Durante a execução do aplicativo:

Todos os controles e os componentes TFindDialog e TReplaceDialog.

LeftAxis

Descrição

A propriedade RightAxis é declarada como um objeto da classe TChartAxis, e representa o eixo vertical esquerdo do gráfico exibido no componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart
```

Durante a execução do aplicativo:

```
TChart, TDBChart
```

LeftCol

Descrição

A propriedade LeftCol é declarada como uma variável inteira que define a coluna que deve ser exibida do lado esquerdo da grade (sem alterar as colunas definidas como fixas). Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Se você quiser que a terceira coluna de um componente StringGrid1 do tipo TStingGrid seja posicionada do lado esquerdo da grade, basta incluir uma linha de código como:

```
StringGrid1.LeftCol := 2;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDrawGrid e TStringGrid
```

LeftMarginInches

Descrição

A propriedade LeftMarginInches é uma variável inteira que define, em décimos de polegada, o valor da margem esquerda do relatório (sem considerar o espaço deixado pela impressora).

Exemplo

Você pode alterar a propriedade Max diretamente no Object Inspector ou mediante uma linha de código como:

```
QuickReport1.LeftMarginInches:= 10;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TQuickReport
```

Durante a execução do aplicativo:

```
TQuickReport
```

LeftMarginMM

Descrição

A propriedade LeftMarginMM é uma variável inteira que define, em milímetros, o valor da margem esquerda do relatório (sem considerar o espaço deixado pela impressora).

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante uma linha de código como:

```
QuickReport1.LeftMarginMM:= 10;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TQuickReport
```

Durante a execução do aplicativo:

```
TQuickReport
```

Length

Descrição

A propriedade Length é declarada como uma variável do tipo inteiro longo (Longint) que define o comprimento do meio a ser executado em um dispositivo multimídia. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir exibe uma mensagem com o comprimento do meio a ser executado em dispositivo multimídia:

```
ShowMessage('Comprimento = '+IntToStr(MediaPlayer1.Length));
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

Level

Descrição

A propriedade Level é declarada como uma variável do tipo Word, e define o nível de um item dentro do componente TOutline que o contém. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TOutlineNode
```

Lines

Descrição

Para componentes do tipo TMemo, essa propriedade é declarada como uma variável do tipo TStrings que contém as linhas de texto do componente.

Para componentes dos tipos TDDEClientItem e TDDEServerItem, essa propriedade é declarada como uma variável do tipo TStrings que contém as linhas do texto a ser transferido numa conversação DDE.

Exemplo

Você pode definir a propriedade Lines diretamente no Object Inspector ou mediante um trecho de programa.

No trecho de código a seguir, o arquivo config.sys é exibido em um componente Memo1 do tipo TMemo quando o usuário dá um clique com o mouse sobre um botão chamado Button1 do tipo TButton:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
      Memo1.Lines.LoadFromFile('C:\CONFIG.SYS');
end;
```

LoadFromFile é um método de Lines, que é um objeto do tipo Tstrings, conforme descrito anteriormente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDDEClientItem, TDDEServerItem, TDBMemo e TMemo
```

Durante a execução do aplicativo:

```
TDDEClientItem, TDDEServerItem, TDBMemo e TMemo
```

LINKBAND

Descrição

Essa propriedade é declarada como uma variável do tipo TQRBand que permite que mais do que um componente do tipo TQRBand seja exibido em uma única página.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRBand
```

Durante a execução do aplicativo:

```
TQRBand
```

LIST

Descrição

A propriedade List é declarada como uma variável do tipo PPointerList que mantém uma lista de ponteiros que pode referenciar objetos de qualquer tipo. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TList
```

LISTFIELD

Descrição

A propriedade ListField é declarada como uma variável do tipo string, e define o nome do campo do banco de dados cujos valor é exibido no controle. Se não for especificado um valor para essa propriedade, será usado o valor armazenado na propriedade KeyField.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBLookupListBox e TDBLookupComboBox
```

Durante a execução do aplicativo:

```
TDBLookupListBox e TDBLookupComboBox
```

ListSource

Descrição

A propriedade ListSource é declarada como uma variável do tipo TDataSource, e define o nome do controle que faz a conexão ao componente que representa o banco de dados do qual será selecionado um campo cujo valor será exibido pelo controle.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBLookupListBox e TDBLookupComboBox
```

Durante a execução do aplicativo:

```
TDBLookupListBox e TDBLookupComboBox
```

Local

Descrição

A propriedade Local é uma variável booleana que define se a tabela referenciada pelo componente é uma tabela local do tipo Paradox ou dBASE ou de um servidor SQL. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TQuery, TDecisionQuery
```

Locale

Descrição

Essa propriedade é declarada como uma variável do tipo TLocale e indica o driver de linguagem usado pelo componente. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDatabase e TSession
```

LocalIP

Descrição

Essa propriedade é declarada como uma variável do tipo string, e define o endereço IP da máquina cliente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TNMEcho, TNMFinger, TNMFTP, TNMHTTP, TNMMSG, TNMMSGServ, TNMNNTP, TNMPOP3, TNMSMTP, TNMSTRM,
TNMSTRMServ e TPowerSock
```

Locked

Descrição

A propriedade Locked é declarada como uma variável booleana que define se um componente do tipo TPanel pode ser substituído por um objeto OLE.

Exemplo

O valor dessa propriedade pode ser definido diretamente no Object Inspector ou mediante uma linha de código como:

```
Panel1.Locked := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPanel
```

Durante a execução do aplicativo:

```
TPanel
```

LoginPrompt

Descrição

A propriedade LoginPrompt é declarada como uma variável booleana que define como é controlado o acesso a um banco de dados SQL.

Exemplo

O valor dessa propriedade pode ser definido diretamente no Object Inspector ou mediante uma linha de código como:

```
Database1.LoginPrompt := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOConnection, TIBDatabase, Tdatabase, TRDSConnection
```

Durante a execução do aplicativo:

```
TADOConnection, TIBDatabase, TDatabase
```

LookupDisplay

Descrição

Essa propriedade é declarada como uma variável string que define o campo do banco de dados a ser exibido pelo componente.

Exemplo

O valor dessa propriedade pode ser definido diretamente no Object Inspector. O trecho de código a seguir faz com que o campo 'Nome' seja exibido pelo componente DBLookupCombo1 do tipo TDBLookupCombo.

```
DBLookupCombo1.LookupDisplay := 'Nome';
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBLookupCombo e TDBLookupList
```

LookupField

Descrição

Essa propriedade é declarada como uma variável do tipo string que faz a ligação entre o banco de dados e o componente.

Exemplo

O valor dessa propriedade pode ser definido diretamente no Object Inspector. O trecho de código a seguir faz com que um componente chamado DBLookupCombo1 do tipo TDBLookupCombo use o campo 'Cliente' como campo de busca:

```
DBLookupCombo1.LookupField := 'Cliente';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBLookupCombo e TDBLookupList
```

Durante a execução do aplicativo:

```
TDBLookupCombo e TDBLookupList
```

LookupSource

Descrição

Essa propriedade é declarada como uma variável do tipo TDataSource que identifica o componente do tipo TDataSource a que está associado o banco de dados no qual deve ser feita a pesquisa.

Exemplo

O valor dessa propriedade pode ser definido diretamente no Object Inspector. O trecho de código a seguir faz com que um componente chamado DBLookupCombo1 do tipo TDBLookupCombo seja associado ao banco de dados especificado no componente DataSource1 do tipo TDataSource:

```
DBLookupCombo1.LookupSource := DataSource1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBLookupCombo e TDBLookupList
```

Durante a execução do aplicativo:

```
TDBLookupCombo e TDBLookupList
```

MainForm

Descrição

A propriedade MainForm é declarada como uma variável do tipo TForm que define o formulário principal de uma aplicação. Esse formulário é o primeiro a ser criado quando a aplicação é executada, e quando ele é fechado, finaliza a execução da aplicação. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir mostra uma mensagem com o texto exibido na barra de títulos da janela principal de uma aplicação.

```
ShowMessage(Application.MainForm.Caption);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TApplication
```

MAPPINGS

Descrição

Essa propriedade é declarada como uma variável do tipo TStrings, e define o modo de mapeamento de colunas entre a tabela-fonte e a tabela de destino durante a execução do método Execute de um componente do tipo TBatchMove.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector usando o String List Editor ou mediante a inclusão de uma linha de código como:

```
DestColName = SourceColName;
```

que mapeia SourceColName da tabela-fonte em DestColName da tabela-destino.

Componentes aos quais se aplica:

Na fase de projeto:

```
TBatchMove
```

Durante a execução do aplicativo:

```
TBatchMove
```

MARGIN

Descrição

Para componentes dos tipos TBitBtn e TSpeedButton , a propriedade Margin é uma variável inteira que define, em pixels, a distância entre um dos lados do bitmap e o lado correspondente do botão que o contém, dependendo da propriedade Layout do botão.

Para componentes do tipo TControlScrollBar, a propriedade Margin é uma variável do tipo Word que define o número mínimo de pixels que deve existir entre os controles de um formulário e suas bordas ou entre os controles de um componente do tipo TScrollBox e suas bordas. É importante ressaltar que, nesse caso, Margin é uma propriedade de um componente do tipo TScrollBar, e que este está embutido em componentes dos tipos TForm e TScrollBox. Nesse caso, o valor da propriedade Margin é automaticamente adicionado ao valor armazenado na propriedade Range, de forma que a barra de rolagem se torne visível sempre que a distância entre o controle e as laterais de um componente do tipo TForm ou TScrollBox torne-se menor que o valor da propriedade Margin.

Exemplo

Para alterar a propriedade Margin de um botão chamado BitBtn1 do tipo TBitBtn ou TSpeedButton durante a execução de um aplicativo, basta incluir a seguinte linha de código no evento correspondente:

```
BitBtn1.Margin := valor;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn, TSpeedButton, TControlScrollBar e TScrollBar
```

Durante a execução do aplicativo:

```
TBitBtn, TSpeedButton, TControlScrollBar e TScrollBar
```

MarginBotton

Descrição

Essa propriedade é declarada como uma variável inteira e define a dimensão da margem inferior do gráfico ou relatório associado ao componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart, TRvNDRWriter
```

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TChart, TDBChart, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

MarginLeft

Descrição

Essa propriedade é declarada como uma variável inteira e define a dimensão, da margem esquerda do gráfico ou relatório associado ao componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart, TRvNDRWriter
```

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TChart, TDBChart, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

MarginRight

Descrição

Essa propriedade é declarada como uma variável inteira e define a dimensão, da margem direita do gráfico ou relatório associado ao componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart, TRvNDRWriter
```

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TChart, TDBChart, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

MarginTop

Descrição

Essa propriedade é declarada como uma variável inteira e define a dimensão da margem superior do gráfico ou relatório associado ao componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart, TRvNDRWriter
```

Durante a execução do aplicativo:

```
TBaseReport, TCanvasReport, TChart, TDBChart, TRvNDRWriter, TRvRenderPreview, TRvRenderPrinter
```

Mask

Descrição

A propriedade Mask é uma variável do tipo string que define um filtro para os arquivos a serem exibidos pelo componente. Podem ser incluídos caracteres curinga como '*' e '?'.

Exemplo

Você pode alterar a propriedade Mask diretamente no Object Inspector ou mediante uma linha de código como:

```
FileListBox1.Mask := '*.PAS';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TFileListBox
```

Durante a execução do aplicativo:

```
TFileListBox e TFilterComboBox
```

Masked

Descrição

A propriedade Masked é declarada como uma variável booleana, que define se as imagens contidas serão ou não transparentes.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
ImageList1,Masked:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TImageList
```

Durante a execução do aplicativo:

```
TImageList
```

Master

Descrição

A propriedade Master é declarada como uma variável do tipo TQRController, que especifica a tabela principal num relacionamento entre duas tabelas, onde uma é a principal e a outra, a secundária.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRDetailLink
```

Durante a execução do aplicativo:

```
TQRDetailLink
```

MasterFields

Descrição

A propriedade MasterFields é declarada como uma variável do tipo string que define os campos que farão a ligação entre duas tabelas. Os nomes dos campos devem ser separados por ponto-e-vírgula.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOTable, TIBTable, TSQLTable, TTable, TClientDataSet, TIBClientDataset, TSimpleDataset
```

Durante a execução do aplicativo:

```
TADOTable, TIBTable, TSQLTable, TTable, TClientDataSet, TIBClientDataset, TSimpleDataset
```

MasterSource

Descrição

A propriedade MasterSource é declarada como uma variável do tipo TDataSource e especifica a tabela-mestre ao qual o componente será ligado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOTable, TIBTable, TSQLTable, TTable, TClientDataSet, TIBClientDataset, TSimpleDataset
```

Durante a execução do aplicativo:

```
TADOTable, TIBTable, TSQLTable, TTable, TClientDataSet, TIBClientDataset, TSimpleDataset
```

Max

Descrição

A propriedade Max é uma variável inteira que define o valor máximo de posição para um componente.

Exemplo

Você pode alterar a propriedade Max diretamente no Object Inspector ou mediante uma linha de código como:

```
ScrollBar1.Max:= 1000;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TProgressBar, TTrackBar, TUpDown e TScrollBar
```

Durante a execução do aplicativo:

```
TProgressBar, TTrackBar, TUpDown e TScrollBar
```

MaxCells

Descrição

Essa propriedade é uma variável inteira que define o número máximo de células manipuladas pelo cache do componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDecisionCube
```

Durante a execução do aplicativo:

```
TDecisionCube
```

MaxFontSize

Descrição

A propriedade MaxFontSize é uma variável inteira que define o maior tamanho da fonte disponível na caixa de diálogo. Um valor nulo significa que não existe valor máximo.

Exemplo

Você pode alterar a propriedade MaxFontSize diretamente no Object Inspector ou mediante uma linha de código como:

```
FontDialog1.MaxFontSize := 36;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TFontDialog
```

Durante a execução do aplicativo:

```
TFontDialog
```

MaxLength

Descrição

A propriedade MaxLength é uma variável inteira que define o número máximo de caracteres que podem ser digitados em um componente dos tipos TEdit, TDBEdit, TMaskEdit, TRichEdit, TDBMemo, TMemo, TDBLookupComboBox ou TComboBox.

Exemplo

Você pode alterar a propriedade MaxLength de um componente diretamente no Object Inspector ou mediante a inclusão de uma linha de código:

```
Edit1.MaxLength:= 80;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TComboBox, TDBEdit, TDBLookupCombo, TDBMemo, TEdit, TMaskEdit, TRichEdit e TMemo
```

Durante a execução do aplicativo:

```
TComboBox, TDBEdit, TDBLookupCombo, TDBMemo, TEdit, TMaskEdit, TRichEdit e TMemo
```

MaxPage

Descrição

A propriedade MaxPage é uma variável inteira que define o maior número para uma

página a ser impressa.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector, ou mediante a inclusão de uma linha de código como:

```
PrintDialog1.MaxPage := 1000;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPrintDialog
```

Durante a execução do aplicativo:

```
TPrintDialog
```

MaxSummaries

Descrição

Essa propriedade é uma variável inteira que define o número máximo de sumários de dados manipuladas pelo cache do componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDecisionCube
```

Durante a execução do aplicativo:

```
TDecisionCube
```

MaxValue

Descrição

A propriedade MaxValue é declarada como uma variável do tipo inteiro longo (Longint) que define o valor máximo que pode ser atribuído a um campo. Atribuir um valor maior do que MaxValue gera uma exceção. Essa propriedade só está disponível durante a execução de um aplicativo.

Exemplo

Você pode alterar o valor da propriedade MaxValue mediante uma linha de código como:

```
IntegerField1.MaxValue:= 1000;
```

onde IntegerField1 é uma variável do tipo TIntegerField.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TCurrencyField, TFloatField, TIntegerField, TSmallintField e TWordField
```

MDIChildCount

Descrição

A propriedade MDIChildCount é uma variável inteira que define o número de janelas-filhas de uma aplicação MDI. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Você pode exibir o valor da propriedade MDIChildCount em um componente Label1 do tipo TLabel mediante a inclusão de uma linha de código:

```
Label1.Caption := Form1.MDIChildCount;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TForm
```

MDIChildren

Descrição

A propriedade MDIChildren é declarada como uma array que contém todas as janelas-filhas de um formulário MDI. Essa propriedade só pode ser acessada durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Se você quiser que um componente chamado Label1 tenha a sua propriedade Caption igual à da terceira janela-filha de um formulário MDI chamado Form1, basta digitar a seguinte linha de código:

```
Label1.Caption := TButton(Form1.MDIChildren[2]).Caption;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TForm
```

Menu

Descrição

A propriedade Menu é declarada como uma variável do tipo TMainMenu que identifica o menu de um formulário.

Exemplo

Você pode alterar a propriedade Menu de um formulário diretamente no Object Inspector, ou mediante a inclusão de uma linha de código como:

```
Form1.Menu := MainMenu1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TForm
```

Message

Descrição

A propriedade Message é declarada como uma variável do tipo string, define o texto a ser exibido na caixa de diálogo que é mostrada quando ocorre um erro definido por essa exceção.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
EAbort, EAbstractError, EAccessViolation, EarrayError, EassertinFailed, EbitsError,
EbrokerException, EcacheError, EclassNotFound, EcommomCalendarError, EcomponentError,
EcontrolC, EconvertError, EDatabaseError, EdateTimeError, EDBClient, EDBEditError,
EDBEngineError, EdimensionMapError, EDimIndexError, EdivByZero, EDSWriter, Eexternal,
EexternalException, EFCreateError, EfilerError, EFOpenError, EHeapException, EInOutError,
EInterpreterError, EintError, EIntCastError, EIntOverFlow, EInvalidArgument, EInvalidCast,
EInvalidGraphic, EInvalidGraphicOperation, EinvalidGridOperation, EinvalidImage, EinvalidOp,
EinvalidOperation, EinvalidPointer, ElistError, ElowCapacityError, EmathError,
EMCIDeviceError, EmenuError, EmonthCalError, EnoResultError, EoleCtrlError, EOLEError,
EoleException, EOleSysError, EoutLineError, EoutOfMemory, EoutOfResources, Eoverflow,
EpackageError, EparserError, Eprinter, Eprivilege, EpropertyError, EpropReadOnly,
```

```
EpropWriteOnky, ErangeError, EradError, EreconcileError, EregistryError, EresNotFound,
EsocketConnectionError, EsocketError, EstackOverflow, EstreamError, EstringListError, Ethread,
EtreeViewError, EunderFlow, EunSupportedTypeError, EUpdateError, EVariantError, EWin32Error,
EWriteError, Exception, EzeroDivide
```

Metafile

Descrição

Essa propriedade é declarada como uma variável do tipo TMetafile, e define o gráfico armazenado no componente. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TPicture
```

Min

Descrição

A propriedade Min é uma variável inteira que define o valor mínimo de posição para um componente.

Exemplo

Você pode alterar a propriedade Min diretamente no Object Inspector ou mediante uma linha de código como:

```
ScrollBar1.Min:= 10;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TProgressBar, TTrackBar, TUpDown e TScrollBar
```

Durante a execução do aplicativo:

```
TProgressBar, TTrackBar, TUpDown e TScrollBar
```

MinFontSize

Descrição

A propriedade MinFontSize é uma variável inteira que define o menor tamanho da fonte disponível na caixa de diálogo. Um valor nulo significa que não existe valor mínimo.

Exemplo

Você pode alterar a propriedade MinFontSize diretamente no Object Inspector ou mediante uma linha de código como:

```
FontDialog1.MinFontSize := 6;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TFontDialog
```

Durante a execução do aplicativo:

```
TFontDialog
```

MinPage

Descrição

A propriedade MinPage é uma variável inteira que define o menor número para uma página a ser impressa.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector, ou mediante a inclusão de uma linha de código como:

```
PrintDialog1.MinPage := 1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPrintDialog
```

Durante a execução do aplicativo:

```
TPrintDialog
```

MinValue

Descrição

A propriedade MinValue é declarada como uma variável do tipo inteiro longo (Longint) que define o valor mínimo que pode ser atribuído a um campo. Atribuir um valor menor do que MinValue gera uma exceção. Essa propriedade só está disponível durante a execução de um aplicativo.

Exemplo

Você pode alterar o valor da propriedade MinValue mediante uma linha de código como:

```
IntegerField1.MinValue:= 1;
```

onde IntegerField1 é uma variável do tipo TIntegerField.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TCurrencyField, TFloatField, TIntegerField, TSmallintField e TWordField
```

ModalResult

Descrição

A propriedade ModalResult é uma variável do tipo TModalResult usada para encerrar a execução de um formulário Modal. Definir o valor da propriedade ModalResult com um valor diferente de zero encerra a execução de um diálogo modal. Normalmente, isto é feito colocando-se botões do tipo TButton ou TBitBtn no formulário e definindo o valor da sua propriedade ModalResult como um dos valores da tabela abaixo. O valor da propriedade ModalResult será o valor retornado pelo método ShowModal que exibe um formulário modal.

Tabela de Valores:

Valor	Significado
mrNone	0
mrOk	idOK (1)
mrCancel	idCancel (2)

continua

continuação

Valor	Significado
mrAbort	idAbort (3)
mrRetry	idRetry (4)
mrIgnore	idIgnore (5)
mrYes	idYes (6)
mrNo	idNo (7)
mrAll	mrNo + 1 (8)

Exemplo

Crie dois formulários chamados Form1 e Form2, cada um com um botão do tipo TButton;

No evento OnClick do botão Button1 de Form1 inclua a seguinte linha de código, e inclua Unit2 na cláusula **Uses** de Form1:

```
Form2.ShowModal;
```

No evento OnClick do botão Button1 de Form2 inclua a seguinte linha de código:

```
ModalResult := 4;
```

Rode o aplicativo e dê um clique no o botão Button1 de Form1, para exibir Form2. Ao clicar no botão Button1 de Form2, este se fechará, pois o valor da sua propriedade Modal Result é diferente de zero. Faça agora a seguinte modificação:

No evento OnClick do botão Button1 de Form2 inclua a seguinte linha de código:

```
ModalResult := 0;
```

Rode o aplicativo e dê um clique no o botão Button1 de Form1, para exibir Form2. Ao clicar no botão Button1 de Form2 este não se fechará, pois o valor da sua propriedade Modal Result é igual a zero.

Observação: A melhor maneira de se trabalhar com um formulário modal é através da inclusão de botões do tipo TButton ou TBitBtn e definir diretamente a propriedade ModalResult dos botões.

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm, TBitBtn e TSpeedButton
```

Durante a execução do aplicativo:

```
TForm, TBitBtn e TSpeedButton
```

Mode

Descrição

A propriedade Mode é declarada como uma variável do tipo TMPModes que define o estado em que se encontra um dispositivo multimídia. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Para componentes do tipo TBatchMove, essa propriedade é declarada como uma variável do tipo TBatchMode, e define o tipo de operação a ser realizada pelo componente.

Para objetos do tipo TPen, define como as linhas serão desenhadas sobre um objeto do tipo Tcanvas.

Tabela de Valores para objetos do Tipo TPen:

Valor	Significado
pmBlack	Sempre preto.
pmWhite	Sempre branco.
pmNop	Transparente.
pmNot	Cor inversa à da tela.
pmCopy	Cor especificada na propriedade Color.
pmNotCopy	Cor inversa da especificada na propriedade Color.
pmMergePenNot	Combinação da cor definida na propriedade Color e a inversa da existente na tela.
pmMaskPenNot	Combinação das cores comuns à cor atual com a inversa da existente na tela.
pmMergeNotPen	Combinação da cor existente na tela com a inversa da cor atual.
pmMaskNotPen	Combinação das cores comuns à existente na tela e a inversa da cor atual.
pmMerge	Combinação da cor atual com a existente na tela.
pmNotMerge	Inverso de pmMerge.
pmMask	Combinação das cores comuns à existente na tela e a cor atual.
pmNotMask	Inverso de pmMask.
pmXor	Combinação das cores comuns na tela e a cor atual, mas não em ambas.
pmNotXor	Inverso de pmXor.

Tabela de Valores para componentes do Tipo TBatchMove:

Valor	Significado
batAppend	Adiciona os registros numa tabela já existente, definida como destino da operação.
batUpdate	Atualiza os registros na tabela definida como destino da operação.
batAppendUpdate	Se um registro correspondente existe na tabela-destino, ele é atualizado,; se não é adicionado um novo registro.
batCopy	Cria a tabela-destino com a mesma estrutura da tabela de origem.
batDelete	Deleta os registros da tabela-destino que têm correspondência com os registros da tabela de origem.

Tabela de Valores para componentes do Tipo TMediaPlayer:

Valor	Significado
mpNotReady	Não está pronto.
mpStopped	Execução terminada.
mpPlaying	Em execução.

continua

continuação

Valor	Significado
mpRecording	Gravando.
mpSeeking	Procurando.
mpPaused	Pausa.
mpOpen	Aberto

Exemplo

O trecho de código a seguir exibe uma mensagem informando se um dispositivo está pronto:

```
if MediaPlayer1.Mode = mpNotReady then ShowMessage('Dispositivo não está pronto')
    else ShowMessage('Dispositivo está pronto');
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBatchMove
```

Durante a execução do aplicativo:

```
TBatchMove e TMediaPlayer
```

Modified

Descrição

Para componentes dos tipos TEdit, TMaskEdit e TMemo, a propriedade Modified é uma variável booleana que define se o texto do componente foi alterado desde que ele foi criado, ou desde a última vez que essa propriedade teve seu valor igual a False. Essa propriedade só está disponível durante a execução do aplicativo.

Para componentes dos tipos TClientDataset, TTable, TQuery, TDecisionQuery e TStoredProc, determina se o registro corrente foi alterado.

Exemplo

Você pode verificar se o texto exibido pelo controle foi alterado durante a execução do aplicativo com a seguinte linha de código:

```
if Edit1.Modified = True then MessageDlg('texto modificado',mtInformation, [mbOK], 0);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TClientDataset, TEdit, TMaskEdit, TTable, TQuery, TDecisionQuery, TStoredProc e TMemo
```

Modifiers

Descrição

A propriedade Modifiers é declarada como uma variável do tipo THKModifiers, e define a tecla modificadora na definição de uma tecla aceleradora.

Tabela de Valores:

Valor	Significado
hkShift	A tecla Shift é usada como tecla modificadora.
hkCtrl	A tecla Ctrl é usada como tecla modificadora.
hkAlt	A tecla Alt é usada como tecla modificadora.
hkExt	Outra tecla é usada como tecla modificadora.

Componentes aos quais se aplica:

Na fase de projeto:

```
THotKey
```

Durante a execução do aplicativo:

```
THotKey
```

ModifySQL

Descrição

Essa propriedade armazena uma lista de strings contendo a declaração SQL usada para modificar um registro.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataset, TIBUpdateSQL, TUpdateSQL
```

Durante a execução do aplicativo:

```
TIBDataset, TIBUpdateSQL, TUpdateSQL
```

Monochrome

Descrição

A propriedade Monochrome é declarada como uma variável booleana, e determina se o objeto contém ou não um gráfico monocromático. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode verificar o valor dessa propriedade durante a execução do aplicativo incluindo uma linha de código como:

```
if Graphic1.Monochrome then ShowMessage('Monocromático');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBitmap
```

MovedCount

Descrição

A propriedade MovedCount é declarada como uma variável do tipo inteiro longo (Longint) que define o número de registros realmente processados pelo método Execute de um componente do tipo TBatchMove. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um controle chamado Label1 do tipo Tlabel exiba o número de registros realmente processados em um componente do tipo TBatchMove:

```
Label1.Caption := IntToStr(BatchMove1.MovedCount);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBatchMove
```

Multiline

Descrição

A propriedade Multiline é declarada como uma variável booleana, que define se as guias de um controle podem ser dispostas em mais de uma linha.

Componentes aos quais se aplica:

Na fase de projeto:

```
TPageControl e TTabControl
```

Durante a execução do aplicativo:

```
TPageControl e TTabControl
```

MultiSelect

Descrição

A propriedade MultiSelect é uma variável booleana que define é possível selecionar simultaneamente mais de um arquivo no componente.

Exemplo

Você pode alterar a propriedade MultiSelect diretamente no Object Inspector ou mediante uma linha de código como:

```
FileListBox1.MultiSelect:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TFileListBox, TListView e TListBox
```

Durante a execução do aplicativo:

```
TFileListBox, TListView e TListBox
```

Name

Descrição

A propriedade Name é uma variável do tipo TComponentName (que é, na realidade, uma string) que identifica o componente dentro da sua aplicação.

Para objetos do tipo TFont, define o nome da fonte utilizada.

Para objetos da classe EcorbaException, identifica o tipo da exceção gerada.

Exemplo

Para alterar a propriedade Name de um botão chamado Button1 durante a execução de um aplicativo, basta incluir a seguinte linha de código no evento correspondente:

```
Button1.Name := 'Novo Nome';
```

Embora o Delphi permita que você altere a propriedade Name de um componente durante a execução de um aplicativo, isto deve ser feito com muita cautela, pois pode provocar verdadeiros desastres! A princípio não deve haver razão para se definir a propriedade Name de um componente durante a execução de um aplicativo, a não ser que o componente esteja sendo criado em tempo de execução.

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles e componentes, além de objetos da classe TAction e TMenuItem.

Durante a execução do aplicativo:

Todos os controles e componentes, além de objetos da classe TAction e TMenuItem.

NAMELIST

Descrição

Esta propriedade é declarada como uma lista de strings, e armazena os nomes dos procedimentos armazenados existentes no banco de dados.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBStoredProc
```

NETFILEDIR

Descrição

Essa propriedade é declarada como uma variável do tipo string, e indica o diretório onde está localizado o arquivo de controle PDOXUSRS.EXE. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TSession
```

NOTIFY

Descrição

A propriedade Notify é uma variável booleana que define se a chamada a um de seus métodos deve gerar um evento OnNotify. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar a propriedade Notify mediante uma linha de código como:

```
MediaPlayer1.Notify := True;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

NotifyValue

Descrição

A propriedade NotifyValue é uma variável do tipo TMPNotifyValues que define o resultado do último método executado em um componente do tipo TMediaPlayer. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Tabela de Valores:

Valor	Significado
nvSuccessful	O método foi executado com sucesso.
nvSuperseded	O método foi superposto por outro método.
nvAborted	O método teve a sua execução interrompida pelo usuário.
nvFailure	O método não foi executado com sucesso.

Exemplo

O trecho de código a seguir exibe uma mensagem informando se o último método foi executado com sucesso:

```
if MediaPlayer1.NotifyValue = nvSuccessful then ShowMessage('O método foi executado com sucesso');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

NumGlyphs

Descrição

A propriedade NumGlyphs é uma variável inteira, variando entre 1 e 4, que define o número de imagens armazenadas no gráfico definido pela propriedade Glyph do componente.

Exemplo

Você pode alterar a propriedade NumGlyphs diretamente no Object Inspector ou mediante uma linha de código como:

```
SpeedButton1.NumGlyphs := 1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn e TSpeedButton
```

Durante a execução do aplicativo:

```
TBitBtn e TSpeedButton
```

ObjClass

Descrição

A propriedade ObjClass é uma variável do tipo string que especifica a classe OLE de um objeto. Normalmente, a classe do objeto é o nome da aplicação sem a extensão .EXE.

Exemplo

Você pode alterar a propriedade ObjClass diretamente no Object Inspector ou mediante uma linha de código como:

```
OLEContainer1.ObjClass = 'Figura do PaintBrush';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOLEContainer
```

Durante a execução do aplicativo:

```
TOLEContainer
```

ObjDoc

Descrição

A propriedade ObjDoc é uma variável do tipo string que especifica o arquivo de um objeto OLE. Normalmente, a classe do objeto é o nome do arquivo que contém o objeto OLE.

Exemplo

Você pode alterar a propriedade ObjDoc diretamente no Object Inspector ou mediante uma linha de código como:

```
OLEContainer1.ObjDoc = 'c:\windows\256color.bmp';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOLEContainer
```

Durante a execução do aplicativo:

```
TOLEContainer
```

Objects

Descrição

Essa variável é declarada como uma array de objetos do tipo TObjects, e dá acesso a um objeto de uma lista de objetos associada a uma lista de strings.

Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TStringList e Tstrings
```

ObjItem

Descrição

A propriedade ObjItem é uma variável do tipo string que especifica o item de dado de um objeto OLE.

Exemplo

Você pode alterar a propriedade ObjItem diretamente no Object Inspector ou mediante uma linha de código como:

```
OLEContainer1.ObjItem = 'Item';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOLEContainer
```

Durante a execução do aplicativo:

```
TOLEContainer
```

OEMConvert

Descrição

A propriedade OEMConvert é uma variável booleana que define se um texto do controle é convertido em caracteres OEM.

Exemplo

Você pode alterar a propriedade OEMConvert diretamente no Object Inspector ou mediante uma linha de código como:

```
Edit1. OEMConvert := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TEdit e TMemo
```

Durante a execução do aplicativo:

```
TEdit e TMemo
```

Open

Descrição

Essa propriedade é declarada como uma variável booleana, e indica se o arquivo que armazena o clipe de vídeo exibido pelo componente está carregado na memória.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TClientDataset, TAnimate
```

Operation

Descrição

A propriedade Operation define a operação que o componente realiza sobre um campo nos registros de um banco de dados.

Tabela de Valores:

Valor	Significado
qrcSUM	Calcula a soma do valor armazenado nesse campo em todos os registros.
qrcCOUNT	Conta o número de registros.
qrcAVERAGE	Calcula a média dos valores armazenados nesse campo em todos os registros.
qrcMIN	Retorna o menor dos valores armazenados nesse campo em todos os registros.
qrcMAX	Retorna o menor dos valores armazenados nesse campo em todos os registros.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRDBCalc
```

Durante a execução do aplicativo:

```
TQRDBCalc
```

OPTIONS

Descrição

O significado dessa propriedade depende do componente a que se refere, como pode ser verificado nas tabelas de valores que se seguem:

Tabela de Valores para objetos do tipo TColorDialog:

Nesse caso, a propriedade é uma variável do tipo TColorDialogOptions e consiste de três subpropriedades booleanas, listadas na tabela abaixo.

Valor	Significado
cdFullOpen	Exibe prontamente as opções de cores personalizadas.
cdPreventFullOpen	Desabilita o botão Create Custom Colors (Definir Cores Personalizadas), impedindo que o usuário possa definir suas próprias cores (Só se aplica se cdFullOpen for False).
cdShowHelp	Exibe um botão de Help na caixa de diálogo.

Tabela de Valores para objetos do Tipo TFontDialog:

Nesse caso, a propriedade é uma variável do tipo TFontDialogOptions e consiste das subpropriedades booleanas listadas na tabela abaixo.

Valor	Significado
fdAnsiOnly	Se igual a True, só permite que o usuário possa selecionar fontes que usam o conjunto de caracteres do Windows, isto é, não permite ao usuário selecionar um estilo de fonte que só contenha símbolos, pois estes não serão exibidos na caixa combo Font.
fdEffects	Seu valor define se as caixas de verificação e a lista de cores devem aparecer no diálogo.

continua

continuação

Valor	Significado
fdFixedPitchOnly	Se igual a True, apenas as fontes monoespaçadas serão exibidas na caixa combo Font.
fdForceFontExist	Indica se deve ser exibida uma mensagem informando ao usuário que ele digitou um nome de fonte inválido.
fdLimitSize	Se igual a True, os valores de MinFontSize e MaxFontSize limitam o número de fontes disponíveis através da caixa de diálogo.
fdNoFaceSel	Se igual a True, nenhuma fonte é pré-selecionada pela caixa de diálogo.
fdNoOEMFonts	Se igual a True, as fontes vetoriais não são exibidas.
fdScalableOnly	Se igual a True, só são exibidas fontes escaláveis.
fdNoSimulations	Se igual a True, só exibe fontes que não são simuladas pela GDI.
fdNoSizeSel	Se igual a True, nenhum tamanho de fonte é pré-selecionado pela caixa de diálogo.
fdNoStyleSel	Se igual a True, nenhum estilo de fonte é pré-selecionado pela caixa de diálogo.
fdNoVectorFonts	O mesmo que fdNoOEMFonts.
fdShowHelp	Se igual a True, a caixa de diálogo exibe um botão de Help.
fdTrueTypeOnly	Se igual a True, somente fontes True Type são pela caixa de diálogo.
fdWysiwyg	Se igual a True, somente fontes disponíveis na tela e na impressora são exibidas pela caixa de diálogo.

Tabela de Valores para Objetos do Tipo TPrintDialog:

Nesse caso, a propriedade é uma variável do tipo TPrintDialogOptions e consiste das subpropriedades booleanas listadas na tabela abaixo.

Valor	Significado
poHelp	Se igual a True, a caixa de diálogo exibe um botão de Help.
poPageNums	Se igual a True, o botão de rádio Pages é habilitado e o usuário pode definir a faixa de páginas a serem impressas.
poPrintToFile	Se igual a True, a caixa de verificação Print to File é exibida, permitindo que o usuário direcione a impressão para um arquivo e não para a impressora.
poSelection	Se igual a True, o botão de rádio Selections é habilitado e o usuário pode optar por imprimir apenas o texto selecionado.
poWarning	Se igual a True, uma mensagem de advertência aparece quando não houver uma impressora instalada.
poDisablePrintToFile	Se igual a True e a opção poPrintToFile também é True, a caixa de verificação Print To File aparece acinzentada. Se poPrintToFile é False, esta opção não tem nenhum efeito.

Tabela de Valores para Objetos do Tipo TOpenDialog, TOpenPictureDialog e TSaveDialog, TSavePictureDialog:

Nesse caso, a propriedade é uma variável do tipo TOpenDialogOptions e consiste das subpropriedades booleanas listadas na tabela a seguir.

Valor	Significado
ofAllowMultiSelect	Se igual a True, permite a seleção de mais de um arquivo na caixa de listagem File Name.
ofCreatePrompt	Se igual a True, quando o usuário digita um nome de arquivo inexistente xibe uma mensagem de advertência e pergunta se deseja criar um arquivo novo com o nome digitado.
ofExtensionDifferent	Essa opção retorna True se a extensão do arquivo selecionado é igual à extensão default, definida na propriedade DefaultExt.
ofFileMustExist	Se igual a True, quando o usuário digita um nome de arquivo inexistente exibe uma mensagem de advertência informando que o arquivo com o nome especificado não foi encontrado, e pergunta se o caminho (path) e o nome digitados estão corretos, não permitindo a criação de novos arquivos.
ofHideReadOnly	Se igual a True, a caixa de verificação Read Only não é exibida na caixa de diálogo.
ofNoChangeDir	Se igual a True, essa opção faz com que o diretório corrente na abertura da caixa de diálogo seja igual ao definido quando a caixa de diálogo apareceu pela primeira vez, ignorando qualquer alteração feita pelo usuário.
ofNoReadOnlyReturn	Se igual a True, uma mensagem de advertência informa ao usuário que o arquivo é somente para leitura.
ofNoTestFileCreate	Essa opção só se aplica quando o usuário tenta salvar um arquivo em um ponto de uma rede do tipo create-no-modify, que não permite que um arquivo seja aberto mais de uma vez e simultaneamente. Se o seu valor é True, a aplicação não verificará se há proteção contra gravação no disco, se o disco está cheio, se uma porta de um drive está aberta, ou se há proteção contra escrita na rede.
ofNoValidate	Se igual a True, essa opção impede que o usuário digite caracteres inválidos para o nome de um arquivo. Se igual a False e o usuário digitar caracteres inválidos para o nome de um arquivo, surge uma mensagem de advertência.
ofOverwritePrompt	Se igual a True, a aplicação exibe uma mensagem de advertência quando o usuário tenta salvar um arquivo que já existe, permitindo que usuário selecione outro nome para o arquivo ou sobreponha o arquivo já existente.
ofReadOnly	Se igual a True, a caixa de verificação aparece marcada quando a caixa de diálogo é exibido.
ofPathMustExist	Se igual a True, só permite que o usuário digite um path já existente para o arquivo. Se o usuário digitar um path inválido, será exibida uma mensagem de advertência.
ofShareAware	Se igual a True, ignora erros de compartilhamento de arquivos. Se False, exibe uma mensagem de advertência quando isto ocorre.
ofShowHelp	Se igual a True, a caixa de diálogo exibe um botão de Help.

Tabela de Valores para objetos do tipos TFindDialog e TReplaceDialog:

Nesse caso, a propriedade é uma variável do tipo TFindOptions e consiste das subpropriedades booleanas listadas na tabela a seguir.

Valor	Significado
frDisableMatchCase	Se igual a True, a caixa de verificação MatchCase aparece acinzentada, e essa opção não pode ser selecionada pelo usuário.
frDisableUpDown	Se igual a True, os botões de rádio Up e Down são desabilitados e não podem ser selecionados pelo usuário.
frDisableWholeWord	Se igual a True, a caixa de verificação Match Whole Word aparece acinzentada, e essa opção não pode ser selecionada pelo usuário.
frDown	Se igual a True, o botão de rádio Down é pré-selecionado pela caixa de diálogo. Se igual a False, o botão de rádio Up é pré-selecionado pela caixa de diálogo.
frFindNext	Se igual a True, faz com que a string definida na propriedade FindText seja pré-selecionada.
frHideMatchCase	Se igual a True, a caixa de verificação MatchCase não é exibida.
frHideWholeWord	Se igual a True, a caixa de verificação Match Whole Word não é exibida.
frHideUpDown	Se igual a True, os botões de rádio Up e Down não são exibidos.
frMatchCase	Se igual a True, a caixa de verificação MatchCase é pré-selecionada pela caixa de diálogo.
frReplace	Se igual a True, a ocorrência atual da string FindText é substituída pela string armazenada na propriedade ReplaceText Essa propriedade só se aplica a objetos do tipo TReplaceDialog.
frReplaceAll	Se igual a True, toda ocorrência da string FindText é substituída pela string armazenada na propriedade ReplaceText Essa propriedade só se aplica a objetos do tipo TReplaceDialog.
frShowHelp	Se igual a True, a caixa de diálogo exibe um botão de Help.
frWholeWord	Se igual a True, a caixa de verificação Match Whole Word é pré-selecionada pela caixa de diálogo.

Tabela de Valores para objetos do Tipo TOutline:

Nesse caso, a propriedade é uma variável do tipo TOutlineOptions e consiste das subpropriedades booleanas listadas a seguir, que definem como os seus itens serão exibidos.

Valor	Significado
ooDrawTreeRoot	O primeiro item (cujo valor é igual a 1) é conectado ao item-raiz.
ooDrawFocusRect	Se igual a True, um retângulo é desenhado em torno do item selecionado, indicando que ele possui o foco.
ooStretchBitmaps	Se igual a True, os itens de Bitmap são redimensionados para caber no tamanho de um item.

Tabela de Valores para objetos do Tipo TDrawGrid e TStringGrid:

Nesse caso, a propriedade é uma variável dos tipos TGridOptions e consiste das subpropriedades booleanas listadas a seguir.

Valor	Significado
goFixedHorzLine	Se igual a True, exibe linhas horizontais entre linhas de células fixas.
goFixedVertLine	Se igual a True, exibe linhas verticais entre colunas de células fixas.
goHorzLine	Se igual a True, exibe linhas horizontais entre linhas de células.
goVertLine	Se igual a True, exibe linhas verticais entre colunas de células.
goRangeSelect	Se igual a True, e desde que goEditing seja False, permite que o usuário selecione um grupo de células de uma só vez.
goDrawFocusSelected	Se igual a True, a célula que possui o foco possui a mesma cor das outras células de um bloco selecionado. Se igual a False, a célula que possui o foco possui a mesma cor das células que não estão selecionadas, isto é, com a cor definida na propriedade Color.
goRowSizing	Se igual a True, as linhas de células podem ser redimensionadas individualmente, exceto as fixas.
goColSizing	Se igual a True, as colunas de células podem ser redimensionadas individualmente, exceto as fixas.
goRowMoving	Se igual a True, o usuário pode deslocar uma linha de células com o mouse.
goColMoving	Se igual a True, o usuário pode deslocar uma linha de células com o mouse.
goEditing	Se igual a True, o usuário pode editar o texto da grade, mas não pode selecionar um grupo de células de uma só vez.
goAlwaysShowEditor	Se igual a True e goEditing também é True, a grade está no modo de edição automático, e o usuário não precisa pressionar Enter ou F2 antes de editar o conteúdo de uma célula. Se igual a False com goEditing igual a True, o usuário precisa pressionar Enter ou F2 antes de editar o conteúdo de uma célula. Se goEditing for False, essa propriedade não tem efeito.
goTabs	Se igual a True, o usuário pode usar Tab e Shift+Tab para se deslocar pelas colunas da grade.
goRowSelect	Se igual a True, o usuário não pode selecionar uma célula individual, mas toda a linha que contém a célula.
goThumbTracking	Se igual a True, o conteúdo da grade rola simultaneamente quando o usuário movimenta a caixa de rolagem da barra de rolagem correspondente. Se igual a False, o rolamento não é simultâneo, ocorrendo apenas quando o usuário libera a caixa de rolagem.

Tabela de Valores para objetos do Tipo TDBGrid e TIWDBGrid:

Nesse caso, a propriedade é uma variável do tipo TDBGridOptions e consiste das subpropriedades booleanas listadas a seguir.

Valor	Significado
dgEditing	Se igual a True, o usuário pode editar os dados da grade. Se a propriedade ReadOnly também é True, o usuário pode inserir linhas em branco com a tecla Insert, ou, quando estiver na extremidade inferior do componente, usar a tecla de seta para baixo para acrescentar uma linha em branco.

Valor	Significado
dgAlwaysShowEditor	Se igual a True e goEditing também é True, a grade está no modo de edição automático, e o usuário não precisa pressionar Enter ou F2 antes de editar o conteúdo de uma célula. Se igual a False com goEditing igual a True, o usuário precisa pressionar Enter ou F2 antes de editar o conteúdo de uma célula. Se goEditing for False, essa propriedade não tem efeito.
dgTitles	Se igual a True, os títulos das colunas são exibidos.
dgIndicator	Se igual a True, exibe um ponteiro que indica a coluna corrente.
dgColumnResize	Se igual a True, as colunas podem ser redimensionadas.
dgColLines	Se igual a True, exibe linhas entre colunas de células.
dgRowLines	Se igual a True, exibe linhas entre linhas de células.
dgTabs	Se igual a True, o usuário pode usar Tab e Shift+Tab para se deslocar pelas colunas da grade.
dgRowSelect	Se igual a True, o usuário não pode selecionar uma célula individual, mas toda a linha que contém a célula.
dgAlwaysShowSelection	Se igual a True, as células selecionadas permanecem selecionadas quando perdem o foco.
dgConfirmDelete	Se igual a True, uma mensagem de confirmação é exibida quando o usuário tenta deletar uma linha de células com Ctrl+Delete.
dgCancelOnExit	Se igual a True e nenhuma modificação tiver sido feita nas células inseridas pelo usuário, a inserção será cancelada quando a grade perder o foco da aplicação.

Tabela de Valores para objetos do Tipo TDBLookupCombo, TDBLookupList, TIWDBLookupCombo e TIWDBLookupList:

Nesse caso, a propriedade é uma variável do tipo TDBLookupListOptions e consiste das subpropriedades booleanas listadas a seguir.

Valor	Significado
loColLines	Se igual a True, as colunas do controle aparecem separadas por linhas verticais.
loRowLines	Se igual a True, as linhas do controle aparecem separadas por linhas horizontais.
loTitles	Se igual a True, os nomes dos campos são exibidos como títulos das colunas do controle.

Tabela de Valores para objetos do Tipo TIndexDef:

Para esses objetos essa propriedade só está disponível durante a execução do aplicativo, não pode ter o seu valor diretamente alterado pelo usuário e corresponde ao conjunto de características do índice, como ixPrimary, ixUnique, ixDescending, ixNonMaintained e ixCaseInsensitive.

Exemplo

Com exceção de TIndexDef, os valores das subpropriedades da propriedade Options de um componente podem ser alterados diretamente no Object Inspector.

Caso queira alterá-las durante a execução do aplicativo, as subpropriedades devem ser listadas entre colchetes, separadas por vírgulas e, então, o conjunto deve ser atribuído à propriedade Options, como na linha de código seguinte:

```
procedure TForm1.FormDblClick(Sender: TObject);
begin
      FindDialog1.Options := [frShowHelp];
      FindDialog1.Execute;
end;
```

Nesse caso, ao se dar um duplo clique com o mouse sobre um formulário chamado Form1, será exibida uma caixa de diálogo do tipo TFindDialog, denominado FindDialog1, com o botão de rádio Up pré-selecionado e exibindo um botão de Help.

As subpropriedades listadas entre colchetes terão o valor True e as demais o valor False, independentemente do valor que lhes tenha sido atribuído no Object Inspector.

Componentes aos quais se aplica:

Na fase de projeto:

TColorDialog, TDBGrid, TDecisionGrid , TDBLookupCombo, TDBLookupList, TDrawGrid, TFontDialog, TFindDialog, TIWDBGrid, TIWDBLookupCombo, TIWDBLookupList, TOpenDialog, TOpenPictureDialog, TOutline, TPrintDialog, TReplaceDialog, TSaveDialog, TSavePictureDialog e TStringGrid

Durante a execução do aplicativo:

TColorDialog, TDBGrid, TDecisionGrid , TDBLookupCombo, TDBLookupList, TDrawGrid, TFontDialog, TFindDialog, TIndexDef, TIWDBGrid, TIWDBLookupCombo, TIWDBLookupList, TOpenDialog, TOpenPictureDialog, TOutline, TPrintDialog, TReplaceDialog, TSaveDialog, TSavePictureDialog e TStringGrid

ORIENTATION

Descrição

Para componentes do tipo TPrinter, essa propriedade é declarada como uma variável do tipo TPrinterOrientation, e determina se a impressão da página será vertical ou horizontal, e só pode ter o seu valor alterado durante a execução do aplicativo.

Para componentes do tipo TTrackBar, essa propriedade é declarada como uma variável do tipo TTrackBarrOrientation, e determina se o controle será vertical ou horizontal.

Para componentes do tipo TUPDown, essa propriedade é declarada como uma variável do tipo TUDOrientation, e define a orientação das setas exibidas no componente.

Para componentes dos tipos TDBCtrlGrid, essa propriedade é declarada como uma variável do tipo TDBCtrlOrientation, e define se os painéis internos ao controle serão dispostos horizontalmente (um ao lado do outro) ou verticalmente (um sobre o outro).

Para componentes do tipo TQuickReport e TQRPrinter, essa propriedade é declarada como uma variável do tipo TPrinterOrientation, e seleciona a orientação do papel na impressão do relatório.

Tabela de Valores para componentes dos Tipos TPrinter, TQRPrinter e TQuickReport:

Valor	Significado
poPortrait	Página impressa verticalmente.
poLandscape	Página impressa horizontalmente.

Tabela de Valores para componentes do Tipo TTrackBar:

Valor	Significado
tbHorizontal	Controle horizontal.
tbVertical	Controle vertical.

Tabela de Valores para componentes do Tipo TUPDown:

Valor	Significado
udHorizontal	Setas para a esquerda e para a direita.
udVertical	Setas para cima e para baixo.

Tabela de Valores para componentes do Tipo TDBCtrlGrid:

Valor	Significado
roHorizontal	Painéis internos dispostos horizontalmente.
roVertical	Painéis internos dispostos verticalmente.

Exemplo

Para alterar a propriedade Orientation de um componente chamado UpDown1 do tipo TUpDown durante a execução de um aplicativo, basta incluir a seguinte linha de código no evento correspondente:

```
TUpDown1.Orientation:= valor;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPrinter, TTrackBar, TUpDown, TQuickReport e TDBCtrlGrid
```

Durante a execução do aplicativo:

```
TTrackBar, TUpDown, TDBCtrlGrid, TQuickReport, TPrinter e TQRPrinter
```

OriginalException

Descrição

A propriedade OriginalException é definida como um objeto da classe Exception , e indica o objeto que representa o erro ocorrido ao se tentar manipular um componente TProvider.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
EUpdateError
```

OUTLINESTYLE

Descrição

A propriedade OutlineStyle é declarada como uma variável do tipo TOutlineStyle que define como a estrutura de dados é exibida em um componente do tipo TOutlineStyle.

Tabela de Valores:

Valor	Significado
osPictureText	Exibe pastas abertas (com a figura especificada na propriedade PictureOpen), pastas fechadas (com a figura especificada na propriedade PictureClosed), pastas de extremidade (com a figura especificada na propriedade PictureLeaf) e itens de texto (especificado na propriedade Text).
osPlusMinusPictureText	Exibe figuras representando um sinal de menos (especificadas na propriedade PictureMinus), figuras representando um sinal de mais (especificadas na propriedade PicturePlus), pastas abertas, pastas fechadas, pastas de extremidade e itens de texto.
osPlusMinusText	Exibe figuras representando um sinal de menos (especificadas na propriedade PictureMinus), figuras representando um sinal de mais (especificadas na propriedade PicturePlus), e itens de texto.
osText	Exibe apenas itens de texto.
osTreePictureText	Exibe figuras representando a árvore hierárquica, pastas abertas, pastas fechadas, pastas de extremidade e itens de texto.
osTreeText	Exibe figuras representando a árvore hierárquica e itens de texto.

Exemplo

Você pode alterar o valor da propriedade OutlineStyle diretamente no Object Inspector ou mediante uma linha de código como:

```
Outline1.OutlineStyle := osPictureText
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOutline
```

Durante a execução do aplicativo:

```
TOutline
```

OVERLOAD

Descrição

Essa propriedade é declarada como uma variável do tipo Word que define o procedimento a ser executado em um servidor Oracle.

Exemplo

Você pode alterar o valor da propriedade Overload diretamente no Object Inspector ou mediante uma linha de código como:

```
StoredProc1.Overload := 1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TStoredProc
```

Durante a execução do aplicativo:

```
TStoredProc
```

Owner

Descrição

A propriedade Owner é uma variável do tipo TComponent que indica o componente proprietário do componente atual. Essa propriedade só pode ser acessada durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Se quisermos saber qual é o componente proprietário de um botão de rádio chamado RadioButton1, devemos incluir a seguinte linha de código, na qual Comp é um objeto do tipo TComponent:

```
Comp := RadioButton1.Owner;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os componentes.

PageCount

Descrição

Para componentes do tipo TQuickReport, essa propriedade é declarada como uma variável do tipo inteiro longo, e especifica o número de páginas do relatório.

Para componentes do tipo TQRPrinter, essa propriedade é declarada como uma variável do tipo inteiro, e especifica o número de páginas do relatório que está sendo impresso.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TQRPrinter e TQuickReport
```

PageHeight

Descrição

Para componentes do tipo TQuickReport, essa propriedade é declarada como uma variável do tipo inteiro longo, e especifica a altura, em pixels, da página no relatório, obtida do objeto QRPrinter associado. Essa propriedade só está disponível durante a execução do aplicativo.

Para componentes do tipo TPrinter, essa propriedade é declarada como uma variável do tipo inteiro, e especifica a altura da página que está sendo impressa. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor alterado pelo usuário.

Para componentes do tipo TQRPrinter, essa propriedade é declarada como uma variável do tipo inteiro, e especifica a altura da página corrente em pixels. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQuickReport
```

Durante a execução do aplicativo:

```
TPrinter, TQuickReport e TQRPrinter
```

PAGEINDEX

Descrição

Para componentes dos tipos TTabbedNotebook e TNotebook, essa propriedade é definida como uma variável inteira que define o número da página ativa.

Para componentes do tipo TTabSheet, essa propriedade é definida como uma variável inteira que define o número da página ativa no controle TPageControl associado.

Exemplo

Você pode alterar a propriedade PageIndex do componente diretamente no Object Inspector ou mediante uma linha de código, como:

```
Notebook1. PageIndex:= 1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTabSheet, TNotebook e TTabbedNotebook
```

Durante a execução do aplicativo:

```
TTabSheet, TNotebook e TTabbedNotebook
```

PAGENUMBER

Descrição

Para componentes do tipo TQRPreview, a propriedade PageNumber é declarada como uma variável inteira que especifica a página a ser pré-visualizada no componente.

Para componentes do tipo TQuickReport, a propriedade PageNumber é declarada como uma variável do tipo inteiro longo que especifica o número da página corrente. Essa propriedade pode ser acessada em eventos durante a preparação de um relatório.

Para componentes do tipo TPrinter, essa propriedade é declarada como uma variável do tipo inteiro e especifica o número da página corrente em um trabalho de impressão. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor alterado pelo usuário.

Para componentes do tipo TQRPrinter, essa propriedade é declarada como uma variável do tipo inteiro e especifica o número da página disponível para ser visualizada ou copiada para o Clipboard.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRPreview
```

Durante a execução do aplicativo:

```
TPrinter, TQRPrinter, TQuickReport e TQRPreview
```

Pages

Descrição

A propriedade Pages é definida como uma variável do tipo TStrings, que é uma lista de strings em que cada string está associada a uma página do componente. A primeira string está associada à primeira página, a segunda string está associada à segunda página, e assim por diante.

Exemplo

Você pode alterar a propriedade Pages do componente diretamente no Object Inspector, com o String List Editor, ou mediante uma linha de código, como:

```
TabbedNotebook1.Pages[0] := 'Página 1';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TNotebook e TTabbedNotebook
```

Durante a execução do aplicativo:

```
TNotebook e TTabbedNotebook
```

PageSize

Descrição

Esta propriedade é declarada como uma variável do tipo inteiro longo, e define o número de bytes por página configurado para o banco de dados.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDatabaseInfo
```

Durante a execução do aplicativo:

```
TIBDatabaseInfo
```

PageWidth

Descrição

Para componentes do tipo TQuickReport, essa propriedade é declarada como uma variável do tipo inteiro longo e especifica a largura da página no relatório, obtida do objeto QRPrinter associado. Essa propriedade só está disponível durante a execução do aplicativo.

Para componentes do tipo TPrinter, essa propriedade é declarada como uma variável do tipo inteiro e especifica a largura da página que está sendo impressa. Essa propriedade só está disponível durante a execução do aplicativo.

Para componentes do tipo TQRPrinter, essa propriedade é declarada como uma variável do tipo inteiro e especifica a largura da página corrente. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TPrinter, TQRPrinter e TQuickReport
```

PALETTE

Descrição

Essa propriedade é declarada como uma variável do tipo HPalette e define a paleta de cores usada pelo objeto. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir copia a paleta de cores de Form2 para Form1:

```
SelectPalette(Form2.Canvas.Handle, Form1.Canvas.Palette, True);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBitmap
```

PANELBORDER

Descrição

A propriedade PanelBorder é declarada como uma variável do tipo TDBCtrlGridBorder e define o tipo de borda desenhada ao redor de cada um dos seus painéis internos.

Tabela de Valores:

Valor	Significado
gbNone	Painéis sem borda.
gbRaised	Painéis com bordas elevadas.

Exemplo

Você pode alterar o valor da propriedade PanelBorder diretamente no Object Inspector ou mediante a inclusão uma linha de código como:

```
DBCtrlGrid1.PanelBorder:= gbNone;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBCtrlGrid
```

Durante a execução do aplicativo:

```
TDBCtrlGrid
```

PANELCOUNT

Descrição

A propriedade PanelCount é declarada como uma variável inteira que especifica o número de painéis visíveis na grade. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBCtrlGrid
```

PanelHeight

Descrição

A propriedade PanelHeight é declarada como uma variável inteira que especifica a altura, em pixels, de cada painel do componente.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
DBCtrlGrid1.PanelHeight:= valor;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBCtrlGrid
```

Durante a execução do aplicativo:

```
TDBCtrlGrid
```

PanelIndex

Descrição

A propriedade PanelIndex é declarada como uma variável inteira que especifica o número do painel correspondente ao registro corrente. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBCtrlGrid
```

Panels

Descrição

A propriedade Panels é declarada como uma variável do tipo TStatusPanels que dá acesso à caixa de diálogo para edição de painéis de um componente que representa uma barra de status.

Componentes aos quais se aplica:

Na fase de projeto:

```
TStatusBar
```

Durante a execução do aplicativo:

```
TStatusBar
```

PanelWidth

Descrição

A propriedade PanelWidth é declarada como uma variável inteira que especifica a largura, em pixels, de cada painel do componente.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
DBCtrlGrid1.PanelWidth:= valor;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBCtrlGrid
```

Durante a execução do aplicativo:

```
TDBCtrlGrid
```

Paragraph

Descrição

Essa propriedade é declarada como uma variável do tipo TParaAttributes e contém informações sobre a formatação do parágrafo que contém o texto selecionado. Caso não exista um texto selecionado, define as informações do parágrafo correspondente à posição atual do cursor. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TRichEdit
```

ParamBindMode

Descrição

Essa propriedade é declarada como uma variável do tipo TParamBindMode que define como os elementos da array Params estarão relacionados aos parâmetros de procedimentos armazenados em servidores.

Exemplo

Você pode alterar o valor da propriedade ParamBindMode diretamente no Object Inspector ou mediante uma linha de código como:

```
StoredProc1. ParamBindMode := pbByName;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TStoredProc
```

Durante a execução do aplicativo:

```
TStoredProc
```

ParamCheck

Descrição

Essa propriedade é definida como uma variável booleana, e define se a lista de parâmetros do componente deve ser gerada novamente quando a declaração SQL é alterada durante a execução do aplicativo.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOCommand, TADODataset, TADOQuery e TADOStoredProc
```

Durante a execução do aplicativo:

```
TADOCommand, TADODataset, TADOQuery e TADOStoredProc
```

ParamCount

Descrição

Para componentes dos tipos TQuery e TDecisionQuery, essa propriedade é declarada como uma variável do tipo Word e indica o número de elementos na array Params do componente.

Para componentes do tipo TStoredProc, essa propriedade é declarada como uma variável do tipo Word e indica o número de parâmetros de input e output para procedimentos localizados em um servidor.

Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBStoredProc, TIBQuery,TStoredProc, Tquery e TDecisionQuery
```

Parameters

Descrição

Essa propriedade é definida como um objeto da classe TParameters, e define ao parâmetros usados em uma declaração SQL.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOCommand, TADODataset, TADOTable, TADOQuery e TADOStoredProc
```

Durante a execução do aplicativo:

```
TADOCommand, TADODataset, TADOTable, TADOQuery e TADOStoredProc
```

Params

Descrição

Para componentes dos tipos TDatabase, TIBTransaction e TIBDatabase, essa propriedade é declarada como uma lista de strings que armazena os parâmetros definidos para acessar um banco de dados representado pelo componente.

Para componentes dos tipos TIBDataset, TIBQuery, TQuery e TDecisionQuery, essa propriedade é declarada como uma array de itens do tipo TParam, que contém os parâmetros de uma declaração SQL dinâmica.

Para componentes dos tipos TIBStoredproc e TStoredProc, essa propriedade armazena os parâmetros a serem passados para procedimentos localizados em um servidor.

Para componentes do tipo TClientDataSet, essa propriedade é declarada como uma array de itens do tipo TParam, que contém os parâmetros a serem enviados ao servidor.

Componentes aos quais se aplica:

Na fase de projeto:

```
TClientDataset, TIBTransaction, TIBStoredproc, TIBDataset, TIBQuery, TStoredProc e TQuery, TDecisionQuery
```

Durante a execução do aplicativo:

```
TClientDataset, TIBTransaction, TIBStoredproc, TIBDataset, TIBQuery, TStoredProc e TQuery, TDecisionQuery
```

PARENT

Descrição

A propriedade Parent é uma variável do tipo TWinControl que indica o componente-pai do componente atual. Essa propriedade só pode ser acessada durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Para objetos do tipo TOutlineNode, é declarada como uma propriedade do tipo TOutlineNode e define o componente TOutline que o contém.

Exemplo

Se quisermos saber qual é o componente-pai de um botão de rádio chamado RadioButton1, devemos incluir a seguinte linha de código, na qual Comp é um objeto do tipo TWinControl:

```
Comp := RadioButton1.Parent;
```

Não confunda componente-pai com componente proprietário. Se você criar um formulário chamado Form1 com um componente GroupBox chamado GroupBox1 e dentro deste inserir um botão de rádio chamado RadioButton1, então os componentes proprietário e pai de RadioButton1 serão, respectivamente, Form1 e GroupBox1.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os controles e TOutlineNode.

PARENTCHART

Descrição

A propriedade ParentChart define o componente gráfico ao qual a série representada pelo objeto está associada.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChartSeries
```

Durante a execução do aplicativo:

```
TChartSeries
```

PARENTCOLOR

Descrição

A propriedade ParentColor é uma variável do tipo booleana que define se o valor da propriedade Color do componente atual deve ou não ser igual à propriedade Color do seu componente-pai.

Componentes aos quais se aplica:

Na fase de projeto:

```
TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBText, TDBListBox,
TDBLookupCombo, TDBLookupList, TDBMemo, TDBRadioGroup, TDirectoryListBox, TDrawGrid,
TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TGroupBox, TLabel, TListBox, TMaskEdit,
TMemo, TNotebook, TOutline, TPaintBox, TPanel, TRadioButton, TRichEdit, TScrollBox e TStringGrid
```

Durante a execução do aplicativo:

```
TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBText, TDBListBox,
TDBLookupCombo, TDBLookupList, TDBMemo, TDBRadioGroup, TDirectoryListBox, TDrawGrid,
TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TGroupBox, TLabel, TListBox, TMaskEdit,
TMemo, TNotebook, TOutline, TPaintBox, TPanel, TRadioButton, TRichEdit, TScrollBox e TStringGrid
```

ParentCtl3D

Descrição

A propriedade ParentCtl3D é uma variável do tipo booleana que define se o valor da propriedade Ctl3D do componente atual deve ou não ser igual à propriedade Ctl3D do seu componente-pai. Se em um formulário fizermos a propriedade ParentCtl3D de todos os controles igual a True, isso garante que todos os controles inseridos no formulário terão o mesmo aspecto (tridimensional ou bidimensional).

Exemplo

Coloque em um formulário chamado Form1 um botão de rádio chamado RadioButton1 e defina, no Object Inspector, sua propriedade ParentCtl3D como False. Nesse caso, você pode definir para o botão de rádio um aspecto diferente do adotado pelo formulário (que é o seu componente-pai).

Componentes aos quais se aplica:

Na fase de projeto:

```
TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage, TDBLookupCombo,
TDBLookupList, TDBListBox, TDBNavigator, TDBMemo, TDBRadioGroup, TDirectoryListBox, TDrawGrid,
TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TGroupBox, TListBox, TMaskEdit, TMemo,
TNotebook, TOutline, TPanel, TRadioButton, TRichEdit, TScrollBox, TStringGrid, TTrackBar e TTreeView
```

Durante a execução do aplicativo:

```
TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage, TDBLookupCombo,
TDBLookupList, TDBListBox, TDBNavigator, TDBMemo, TDBRadioGroup, TDirectoryListBox, TDrawGrid,
TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TGroupBox, TListBox, TMaskEdit, TMemo,
TNotebook, TOutline, TPanel, TRadioButton, TRichEdit, TScrollBox, TStringGrid, TTrackBar e TTreeView
```

ParentFont

Descrição

A propriedade ParentFont é uma variável do tipo booleana que define se o valor da propriedade Font do componente atual deve ou não ser igual à propriedade Font do seu componente-pai. Se em um formulário fizermos a propriedade ParentFont de todos os controles igual a True, isso garantirá que todos os controles inseridos no formulário usarão o mesmo tipo de fonte na exibição de textos.

Exemplo

Coloque em um formulário Form1 um botão chamado Button1 e defina, no Object Inspector, sua propriedade ParentFont como False. Nesse caso, você pode definir para o botão um estilo de fonte diferente do adotado pelo formulário (que é o seu componente-pai).

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage,
TDBLookupCombo, TDBLookupList, TDBListBox, TDBMemo, TDBRadioGroup, TDBText, TDirectoryListBox,
TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TForm, TGroupBox, THeader,
THeaderControl, TListBox, TMaskEdit, TMemo, TNotebook, TOutline, TPaintBox, TPanel,
TRadioButton, TRichEdit, TScrollBox, TSpeedButton, TStatusBar, TStringGrid e TtreeView
```

Durante a execução do aplicativo:

```
TBitBtn, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage,
TDBLookupCombo, TDBLookupList, TDBListBox, TDBMemo, TDBRadioGroup, TDBText, TDirectoryListBox,
TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TForm, TGroupBox, THeader,
THeaderControl, TListBox, TMaskEdit, TMemo, TNotebook, TOutline, TPaintBox, TPanel,
TRadioButton, TRichEdit, TScrollBox, TSpeedButton, TStatusBar, TStringGrid e TTreeView
```

PARENTSHOWHINT

Descrição

A propriedade ParentShowHint é uma variável do tipo booleana que define se o valor da propriedade ShowHint do controle atual deve ou não ser igual à propriedade ShowHint do seu componente-pai. Seu valor default é True, o que indica que, se em um formulário fizermos a sua propriedade ShowHint igual a False, e todos os controles nele inseridos tiverem a propriedade ParentShowHint igual a True, nenhuma string de auxílio será exibida.

Exemplo

Coloque em um formulário Form1 um botão chamado Button1 e defina, no Object Inspector, sua propriedade Hint como *Botão* e ShowHint como True. Execute o aplicativo e a string de auxílio será exibida quando o mouse estiver sobre o botão. Defina a propriedade ParentShowHint de Button1 como True e execute novamente o aplicativo. Nesse caso a string de auxílio não será exibida.

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles.

Durante a execução do aplicativo:

Todos os controles.

PASSWORDCHAR

Descrição

A propriedade PasswordChar é uma variável do tipo Char que permite a criação de uma caixa de edição que exiba caracteres especiais em vez do texto digitado.

Exemplo

Você pode definir o valor da propriedade PasswordChar de um componente diretamente no Object Inspector, ou mediante uma linha de código:

```
Edit1.PasswordChar := '@';
```

Definir a propriedade PasswordChar igual a '#0' faz com que o texto digitado seja exibido normalmente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBEdit, TEdit e TMaskEdit
```

Durante a execução do aplicativo:

```
TDBEdit, TEdit e TMaskEdit
```

Pen

Descrição

A propriedade Pen é declarada como uma variável do tipo TPen que define o tipo de caneta a ser usada pelo componente.

Exemplo

Você pode alterar as subpropriedades da caneta de um componente diretamente no Object Inspector (apenas controles do tipo TShape) ou mediante a inclusão de uma linha de código como:

```
Shape1.Pen.Width := 40;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TShape
```

Durante a execução do aplicativo:

```
TCanvas e TShape
```

PenPos

Descrição

A propriedade PenPos é declarada como uma variável do tipo TPoint e define a posição corrente da caneta no desenho.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TCanvas
```

Picture

Descrição

A propriedade Picture é declarada como uma variável do tipo TPicture que define a imagem a ser exibida pelo controle.

Exemplo

Você pode alterar o valor da propriedade Picture de um componente Image1 do tipo TImage diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
Image1.Picture.LoadFromFile('BITMAP1.BMP');
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TImage
```

Durante a execução do aplicativo:

```
TDBImage e TImage
```

PictureClosed

Descrição

A propriedade PictureClosed é declarada como uma variável do tipo TBitmap que define a imagem a ser exibida em um componente do tipo TOutline para representar um item que possui subitens mas não está expandido.

Exemplo

Você pode alterar o valor da propriedade PictureClosed de um componente Outline1 do tipo TOutline diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
Outline1.PictureClosed.LoadFromFile('c:\BITMAP1.BMP');
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOutline
```

Durante a execução do aplicativo:

```
TOutline
```

PICTURELEAF

Descrição

A propriedade PictureLeaf é declarada como uma variável do tipo TBitmap que define a imagem a ser exibida em um componente do tipo TOutline para representar um item que não possui subitens.

Exemplo

Você pode alterar o valor da propriedade PictureLeaf de um componente Outline1 do tipo TOutline diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
Outline1.PictureLeaf.LoadFromFile('c:\BITMAP1.BMP');
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOutline
```

Durante a execução do aplicativo:

```
TOutline
```

PICTUREMINUS

Descrição

A propriedade PictureMinus é declarada como uma variável do tipo TBitmap que define a imagem a ser exibida em um componente do tipo TOutline para representar um item que possui subitens e está expandido.

Exemplo

Você pode alterar o valor da propriedade PictureMinus de um componente Outline1 do tipo TOutline diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
Outline1.PictureMinus.LoadFromFile('c:\BITMAP1.BMP');
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOutline
```

Durante a execução do aplicativo:

```
TOutline
```

PictureOpen

Descrição

A propriedade PictureOpen é declarada como uma variável do tipo TBitmap que define a imagem a ser exibida em um componente do tipo TOutline para representar um item que possui subitens e está expandido.

Exemplo

Você pode alterar o valor da propriedade PictureOpen de um componente Outline1 do tipo TOutline diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
Outline1.PictureOpen.LoadFromFile('c:\BITMAP1.BMP');
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOutline
```

Durante a execução do aplicativo:

```
TOutline
```

PicturePlus

Descrição

A propriedade PicturePlus é declarada como uma variável do tipo TBitmap que define a imagem a ser exibida em um componente do tipo TOutline para representar um item que possui subitens mas não está expandido.

Exemplo

Você pode alterar o valor da propriedade PicturePlus de um componente Outline1 do tipo TOutline diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
Outline1.PicturePlus.LoadFromFile('c:\BITMAP1.BMP');
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOutline
```

Durante a execução do aplicativo:

```
TOutline
```

Pixels

Descrição

A propriedade Pixels é declarada como uma variável do tipo TColor e permite que você acesse diretamente um pixel no desenho, para ler ou atribuir uma cor. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode definir como amarela a cor do pixel nas coordenadas (10,10) com a seguinte linha de código:

```
Pixels[10,10] := clYellow;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TCanvas
```

PIXELSPERINCH

Descrição

A propriedade PixelsPerInch é declarada como uma variável inteira que define o número de pixels por polegada em um formulário, no driver de vídeo corrente.

Exemplo

Você pode alterar o valor da propriedade PixelsPerInch diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
Form1.PixelsPerInch := 80;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TForm e TScreen
```

PLAINTEXT

Descrição

Essa propriedade é declarada como uma variável do tipo booleana que define se o texto será exibido com um único tipo de formatação ou com atributos de formatação distintos. Seu valor default é igual a False.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou, durante a execução do aplicativo, mediante a inclusão de uma linha de código como:

```
RichEdit1.PlainText:= False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TRichEdit
```

Durante a execução do aplicativo:

```
TRichEdit
```

POPUPCOMPONENT

Descrição

A propriedade PopupComponent é declarada como uma variável do tipo TComponent que define o último componente selecionado pelo usuário que fez com que o menu Popup representado por esse controle fosse exibido. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TPopupMenu
```

PopupMenu

Descrição

A propriedade PopupMenu é uma variável do tipo TPopupMenu que define o nome do menu flutuante que aparece quando o usuário seleciona um componente e pressiona o botão direito do mouse (desde que a propriedade AutoPopup do menu flutuante seja igual a True) ou quando se executa o método Popup do menu flutuante.

Exemplo

Você pode definir o valor da propriedade PopupMenu de um componente diretamente no Object Inspector, ou mediante uma linha de código:

```
Form1.PopupMenu:= PopupMenu1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage,
TDBLookupCombo, TDBLookupList, TDBListBox, TDBMemo, TDBNavigator, TDBText, TDBRadioGroup,
TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TForm, TGroupBox,
THeaderControl, TImage, TLabel, TListBox, TListView, TMaskEdit, TMemo, TNotebook, TPageControl,
TPanel, TPaintBox, TRadioButton, TRichEdit, TScrollBar, TScrollBox, TStatusBar, TStringGrid,
TTabControl, TTabSheet, TTrackBar e TTreeView
```

Durante a execução do aplicativo:

```
TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage,
TDBLookupCombo, TDBLookupList, TDBListBox, TDBMemo, TDBNavigator, TDBText, TDBRadioGroup,
TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TForm, TGroupBox,
THeaderControl, TImage, TLabel, TListBox, TListView, TMaskEdit, TMemo, TNotebook,
TPageControl, TPanel, TPaintBox, TRadioButton, TRichEdit, TScrollBar, TScrollBox, TStatusBar,
TStringGrid, TTabControl, TTabSheet, TTrackBar e TTreeView
```

Port

Descrição

Essa propriedade é declarada como uma variável inteira e identifica a porta pela qual se efetua a conexão ao servidor.

Componentes aos quais se aplica:

Na fase de projeto:

```
TNMDayTime, TNMEcho, TNMFinger, TNMFTP, TNMHTTP, TNMMSG, TNMMSGServ, TNMNNTP, TNMPOP3,
TNMSMTP, TNMSTRM, TNMSTRMServ, TNMTime, TpowerSock
```

Durante a execução do aplicativo:

```
TNMDayTime, TNMEcho, TNMFinger, TNMFTP, TNMHTTP, TNMMSG, TNMMSGServ, TNMNNTP, TNMPOP3,
TNMSMTP, TNMSTRM, TNMSTRMServ, TNMTime, TPowerSock
```

Position

Descrição

Para componentes do tipo TForm, a propriedade Position é declarada como uma variável do tipo TPosition que define o tamanho e posição de um formulário, no momento em que ele aparece na sua aplicação.

Para componentes do tipo TMediaPlayer, a propriedade Position é declarada como uma variável do tipo inteiro longo (Longint) e define a posição atual durante a execução de um dispositivo multimídia.

Para componentes dos tipos TControlScrollBar e TScrollBar, a propriedade Position é declarada como uma variável inteira que define a posição atual da caixa de rolagem em uma barra de rolamento.

Para componentes dos tipos TFindDialog e TReplaceDialog, a propriedade Position é declarada como uma variável do tipo TPoint que define a posição em que o quadro de diálogo é exibido na tela.

Para componentes do tipo TProgressBar, a propriedade Position é declarada como uma variável inteira que define a posição corrente na execução de uma tarefa.

Para componentes do tipo TTrackBar, a propriedade Position é declarada como uma variável inteira que define a posição corrente indicada pelo controle.

Para componentes do tipo TUpDown, a propriedade Position é declarada como uma variável inteira que define a posição corrente indicada pelo controle.

Tabela de Valores Para componentes do Tipo TForm:

Valor	Significado
poDesigned	O formulário aparece no mesmo tamanho e posição exibidos na fase de projeto.
poDefault	O formulário aparece com o tamanho e posição a serem definidos pelo Delphi.
poDefaultPosOnly	O formulário aparece no mesmo tamanho exibido na fase de projeto, mas a posição é definida pelo Delphi.
poDefaultSizeOnly	O formulário aparece na mesma posição exibida na fase de projeto, mas o tamanho é definido pelo Delphi.
poScreenCenter	O formulário aparece no mesmo tamanho exibido na fase de projeto, mas a posição é sempre no centro da tela.

Exemplo

Você pode definir o valor da propriedade Position de um formulário diretamente no Object Inspector, ou mediante uma linha de código como:

```
Form1.Position := poScreenCenter;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm, TMediaPlayer, TControlScrollBar, TProgressBar, TTrackBar, TUpDown e TScrollBar
```

Durante a execução do aplicativo:

```
TForm, TFindDialog, TMediaPlayer, TControlScrollBar, TProgressBar, TTrackBar, TUpDown,
TReplaceDialog e TScrollBar
```

PostMessage

Descrição

Essa propriedade é declarada como um objeto da classe TPostMessage e contém todas as informações referentes à mensagem a ser enviada por correio eletrônico.

Componentes aos quais se aplica:

Na fase de projeto:

```
TNMSMTP
```

Durante a execução do aplicativo:

```
TNMSMTP
```

Precision

Descrição

A propriedade Precision é declarada como uma variável inteira que define o número de casas decimais a serem exibidas por um campo numérico. Essa propriedade só está disponível durante a execução de um aplicativo.

Exemplo

Você pode alterar o valor da propriedade Precision mediante uma linha de código como:

```
FloatField1.Precision := 2;
```

onde FloatField1 é uma variável do tipo TFloatField.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBCDField, TCurrencyField e TFloatField
```

Prepared

Descrição

Para componentes dos tipos TIBDataset, TIBQuery, TQuery, TDecisionQuery, essa propriedade é declarada como uma variável booleana e define se foi feita uma chamada ao método Prepare do componente.

Para componentes dos tipos TIBStoredProc, TStoredProc, essa propriedade é declarada como uma variável booleana e define se foi feita uma otimização do procedimento armazenado no servidor.

Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADOCommand, TADODataset, TADOTable, TADOQuery,  TADOStoredProc, TIBStoredProc, TIBDataset,
TIBQuery, TStoredProc e TQuery, TDecisionQuery
```

PreviousError

Descrição

Essa propriedade é definida como um a variável inteira e retorna o código do erro anterior ao que gerou a exceção. Caso o erro não esteja relacionado ao BDE, será retornado o valor 0.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
EUpdateError
```

PrintBefore

Descrição

A propriedade PrintBefore é declarada como uma variável booleana que especifica se a tabela secundária deve ser impressa antes da tabela principal.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRDetailLink
```

Durante a execução do aplicativo:

```
TQRDetailLink
```

PrinterIndex

Descrição

A propriedade PrinterIndex é declarada como uma variável do tipo inteiro e retorna o índice da impressora corrente dentre as listadas na propriedade Printers. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TPrinter
```

PrinterOK

Descrição

Essa propriedade é declarada como uma variável booleana e define se um driver de impressão está instalado. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TQRPrinter
```

Printers

Descrição

A propriedade Printers é declarada como uma variável do tipo TStrings, e consiste em uma lista de strings que identificam todas as impressoras instaladas pelo Windows. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Você pode exibir os nomes de todas as impressoras instaladas em um componente ListBox1 do Tipo TListBox mediante a inclusão da seguinte linha de código:

```
ListBox1.Items := Printer1.Printers;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TPrinter
```

Printing

Descrição

A propriedade Printing é declarada como uma variável do tipo booleana e indica se o sistema está executando um trabalho de impressão. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Para componentes TChart e TDBChart, essa propriedade indica se o gráfico exibido no componente está sendo impresso.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TChart, TDBChart e TPrinter
```

PrintMask

Descrição

A propriedade PrintMask é declarada como uma variável do tipo String e define um formato para a exibição do valor armazenado no componente. A formatação será a mesma usada na função FormatFloat.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRDBCalc
```

Durante a execução do aplicativo:

```
TQRDBCalc
```

PrintRange

Descrição

A variável PrintRange é declarada como uma variável do tipo TPrintRange que define o tipo de faixa de páginas a ser usado durante uma impressão.

Tabela de Valores:

Valor	Significado
prAllPages	Todas as páginas são impressas.
prSelection	Imprime apenas o texto ou objeto selecionado.
prPageNums	Permite que o usuário selecione as páginas a serem impressas.

Exemplo

Você pode definir o valor da propriedade PrintRange diretamente no Object Inspector, ou mediante uma linha de código como:

```
PrintDialog1.PrintRange := prAllPages;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPrintDialog
```

Durante a execução do aplicativo:

```
TPrintDialog
```

PrintScale

Descrição

A variável PrintScale é declarada como uma variável do tipo TPrintScale que define as proporções de um formulário impresso.

Tabela de Valores:

Valor	Significado
poNone	Nesse caso, o formulário é impresso praticamente como aparece na tela, podendo haver pequenos ajustes.
poProportional	O formulário é impresso com o mesmo tamanho que aparece na tela (com o mesmo número de pixels por polegada).
poPrintToFit	As dimensões do formulário mantêm a mesma proporção, mas são alteradas por um fator de escala, de forma a preencher a página a ser impressa.

Exemplo

Você pode definir o valor da propriedade PrintScale de um formulário diretamente no Object Inspector ou mediante uma linha de código como:

```
Form1.PrintScale := poNone;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TForm
```

PrintToFile

Descrição

A propriedade PrintToFile é declarada como uma variável booleana que define se a impressão deve ser dirigida para um arquivo.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
PrintDialog1.PrintToFile := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPrintDialog
```

Durante a execução do aplicativo:

```
TPrintDialog
```

PrivateDir

Descrição

Essa propriedade é declarada como uma variável do tipo string e indica o diretório no qual os arquivos temporários devem ser armazenados. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TSession
```

ProblemCount

Descrição

A propriedade ProblemCount é declarada como uma variável do tipo inteiro longo (Longint) que define o número de registros que não foram adicionados à tabela-destino devido a algum erro no processamento do método Execute de um componente do tipo TBatchMove. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um controle chamado Label1 do tipo Tlabel exiba o valor da propriedade ProblemCount de um componente do tipo TBatchMove:

```
Label1.Caption := IntToStr(BatchMove1.ProblemCount);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBatchMove
```

ProblemTableName

Descrição

A propriedade ProblemTableName é declarada como uma variável do tipo TFileName que armazena os registros que não puderam ser movidos devido a um erro em uma operação realizada por um componente do tipo TBatchMove.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou por meio

de uma linha de código, como:

```
BatchMove1.ProblemTableName := 'registro.db';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBatchMove
```

Durante a execução do aplicativo:

```
TBatchMove
```

ProcedureName

Descrição

Essa propriedade é declarada como uma variável do tipo string que define o nome do procedimento a ser executado em um servidor de banco de dados acessado através do mecanismo Activex Data Objects.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOStoredProc
```

Durante a execução do aplicativo:

```
TADOStoredProc
```

ProjectFile

Descrição

Essa propriedade é declarada como uma variável do tipo string que define o nome do arquivo de projeto de relatório criado com o Rave Reports, com a extensão .RAV.

Componentes aos quais se aplica:

Na fase de projeto:

```
TRvProject
```

Durante a execução do aplicativo:

```
TRvProject
```

Provider

Descrição

A propriedade Provider especifica a interface do tipo IProvider por meio da qual esse componente se comunica com um componente TProvider.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TClientDataset
```

ProviderName

Descrição

A propriedade Provider é declarada como uma variável do tipo string e define o nome do componente TProvider por meio do qual é feita a comunicação com o servidor.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TClientDataset
```

QDelete

Descrição

Essa propriedade é declarada como um objeto da classe TIBSQL, e permite acessar diretamente o objeto que armazena o comando SQL para exclusão de registros.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataset
```

Durante a execução do aplicativo:

```
TIBDataset
```

QInsert

Descrição

Essa propriedade é declarada como um objeto da classe TIBSQL, e permite acessar diretamente o objeto que armazena o comando SQL para inclusão de registros.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataset
```

Durante a execução do aplicativo:

```
TIBDataset
```

QModify

Descrição

Essa propriedade é declarada como um objeto da classe TIBSQL, e permite acessar diretamente o objeto que armazena o comando SQL para atualização de registros.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataset
```

Durante a execução do aplicativo:

```
TIBDataset
```

QRefresh

Descrição

Essa propriedade é declarada como um objeto da classe TIBSQL, e permite acessar diretamente o objeto que armazena o comando SQL para atualizar a exibição de registros.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataset
```

Durante a execução do aplicativo:

```
TIBDataset
```

QSelect

Descrição

Essa propriedade é declarada como um objeto da classe TIBSQL, e permite acessar diretamente o objeto que armazena o comando SQL para exibição de registros.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataset
```

Durante a execução do aplicativo:

```
TIBDataset
```

Query

Descrição

Para componentes dos tipos TIBUpdateSQL e TUpdateSQL essa propriedade retorna um componente do tipo TQuery, correspondente à declaração SQL cujo índice é igual a UpdateKind. UpdateKind pode ser igual a ukModify, ukDelete ou ukInsert. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Para componentes do tipo TRvQueryConnection, essa propriedade define o componente do tipo TQuery ao qual está vinculado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TRvQueryConnection
```

Durante a execução do aplicativo:

```
TIBUpdateSQL, TRvQueryConnection, TUpdateSQL
```

Queued

Descrição

A propriedade Queued é declarada como uma variável booleana que indica se há evento pendentes.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBEvents
```

Durante a execução do aplicativo:

```
TIBEvents
```

ReadOnly

Descrição

Para componentes dos tipos TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField, TDateTimeField, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField, TStringField, TTimeField, TVarBytesField, TWordField, TDBMemo, TDBRadioGroup, TEdit, TMaskEdit e TMemo a propriedade ReadOnly é uma variável booleana que define se o usuário pode alterar o valor exibido por um controle. No caso de controles correspondentes a bancos de dados, especifica se o usuário pode alterar o valor armazenado em um campo de um registro; em componentes do tipo TDBGrid, define se o usuário pode inserir linhas e colunas; em componentes do tipo TTable, define se o usuário pode alterar o conteúdo de uma tabela.

Exemplo

Você pode definir o valor da propriedade ReadOnly de um componente diretamente no Object Inspector ou mediante uma linha de código:

```
DBEdit1.ReadOnly:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo,
TDBLookupList, TDBMemo, TDBRadioGroup, TEdit, TMaskEdit, TMemo, TADOTable, TClientDataset,
TIBTable e TTable
```

Durante a execução do aplicativo:

```
TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList,
TDBMemo, TDBRadioGroup, TEdit, TMaskEdit, TMemo, TClientDataset, TADOTable, TClientDataset,
TIBTable, TTable, TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField,
TCurrencyField, TDateField, TDateTimeField, TFloatField, TGraphicField, TIntegerField,
TMemoField, TSmallintField, TStringField, TTimeField, TVarBytesField, TWordField
```

RecordCount

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro longo e especifica o número de registros do banco de dados associado. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataset, TDecisionQuery, TIBDataset,
TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery, TDecisionQuery e TStoredProc
```

RecNo

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro longo e especifica o registro corrente do banco de dados associado. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataset, TDecisionQuery, TIBDataset,
TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery, TDecisionQuery e TStoredProc
```

RecordSize

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro longo e especifica o tamanho em bytes de um registro da tabela associada ao componente. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataset, TDecisionQuery, TIBDataset,
TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery, TDecisionQuery e TStoredProc
```

RDSConnection

Descrição

Essa propriedade é declarada como um objeto da classe TRDSConnection, e indica o componente através do qual será feita a conexão através do mecanismo Activex Data Objects.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADODataset
```

Durante a execução do aplicativo:

```
TADODataset
```

RefreshSQL

Descrição

Essa propriedade armazena uma lista de strings contendo a declaração SQL usada para atualizar o resultado de uma declaração SQL.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataset, TIBUpdateSQL
```

Durante a execução do aplicativo:

```
TIBDataset, TIBUpdateSQL
```

Registered

Descrição

A propriedade Queued é declarada como uma variável booleana que indica se o evento definido pelo componente foi registrado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBEvents
```

Durante a execução do aplicativo:

```
TIBEvents
```

RemoteServer

Descrição

A propriedade RemoteServer define o componente de conexão por meio da qual é feita a comunicação com a aplicação servidora.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TClientDataset
```

Repetitions

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro e define o número de execuções contínuas do clipe de vídeo exibido pelo componente. Atribuir o valor 0 a essa propriedade faz com que o clipe seja executado continuamente, até que se atribua o valor False à sua propriedade Active.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAnimate
```

ReplaceText

Descrição

A propriedade ReplaceText é declarada como uma variável do tipo string e contém o texto que substituirá aquele a ser pesquisado na aplicação.

Exemplo

Você pode definir o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
ReplaceDialog1.ReplaceText := 'Texto substituto';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TReplaceDialog
```

Durante a execução do aplicativo:

```
TReplaceDialog
```

ReportTitle

Descrição

Essa propriedade é declarada como uma variável do tipo string e define o título exibido no gerenciador de impressão do Windows quando o relatório é impresso.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQuickReport
```

Durante a execução do aplicativo:

```
TQuickReport
```

ReportType

Descrição

Essa propriedade é declarada como um conjunto que pode assumir um dos valores citados na tabela a seguir e define o tipo de relatório gerado pelo componente. Essa propriedade só está disponível durante a execução do aplicativo.

Tabela de Valores:

Valor	Significado
qrStandard	Gera um relatório padrão, a partir do banco de dados associado a um componente do tipo TTable ou TQuery, TDecisionQuery.
qrMasterDetail	Gera um relatório do tipo Formulário principal/Formulário secundário, usando componentes do tipo TQRDetailLink.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TQuickReport
```

RequestURL

Descrição

Essa propriedade é declarada como uma variável do tipo string e define a URL do documento carregado no Browser representado pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
THTML
```

Required

Descrição

A propriedade Required é uma variável booleana que define se um campo deve ter obrigatoriamente um valor não-nulo. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar o valor dessa propriedade mediante uma linha de código como:

```
if FloatField1.Required = True then ShowMessage('Requer valor não-nulo');
```

onde FloatField1 é uma variável do tipo TFloatField.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFieldDef, TFloatField, TGraphicField, TIntegerField, TMemoField,
TSmallintField, TStringField, TTimeField, TVarBytesField e TWordField
```

ResetGroup

Descrição

A propriedade ResetGroup é declarada como uma variável do tipo TQRBand e define se o componente será reinicializado após a impressão do valor definido na sua propriedade ResetBand.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRDBCalc
```

Durante a execução do aplicativo:

```
TQRDBCalc
```

ResHandle

Descrição

Essa propriedade é declarada como uma variável do tipo THandle (que é, na realidade, um inteiro) e retorna um Handle do Windows para o módulo que armazena o clipe de vídeo exibido pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAnimate
```

ResId

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro e define o recurso do Windows que armazena o clipe de vídeo exibido pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAnimate
```

ResName

Descrição

Essa propriedade é declarada como uma variável do tipo string e define o nome do arquivo de recurso do Windows que armazena o clipe de vídeo exibido pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAnimate
```

RestartData

Descrição

Essa propriedade é declarada como uma variável booleana e define se a impressão deve ser iniciada no primeiro registro (True) ou no registro corrente (False).

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
QuickReport1.RestartData:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TQuickReport
```

Durante a execução do aplicativo:

```
TQuickReport
```

RightAxis

Descrição

A propriedade RightAxis é declarada como um objeto da classe TChartAxis e representa o eixo vertical direito do gráfico exibido no componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart
```

Durante a execução do aplicativo:

```
TChart, TDBChart
```

ROW

Descrição

A propriedade Row é declarada como uma variável do tipo inteiro longo (Longint) que define a que linha pertence a célula que possui o foco. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe a linha que possui o foco em um componente StringGrid1 do tipo TStringGrid quando o usuário seleciona uma célula com o mouse.

```
procedure TForm1.StringGrid1Click(Sender: TObject);
begin
    Label1.Caption := 'Linha ' + IntToStr(StringGrid1.Row + 1);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDrawGrid, TOutline e TStringGrid
```

ROWCOUNT

Descrição

Para componentes dos tipos TDrawGrid, TIWDBGrid e TStringGrid, essa propriedade é declarada como uma variável do tipo inteiro longo (Longint) que define o número de linhas do controle.

Para esses componentes, essa propriedade só está disponível durante a execução do aplicativo.

Para componentes dos tipos TDBCtrlGrid e TDecisionGrid, essa propriedade é declarada como uma variável do tipo inteiro que define o número de linhas do controle. Essa propriedade só está disponível durante a execução do aplicativo.

Para componentes dos tipos TDBLookupListBox e TIWDBLookupListBox, essa propriedade é declarada como uma variável do tipo inteiro que define o número de linhas exibidas pelo controle.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe o número de linhas de um componente StringGrid1 do tipo TStringGrid quando o usuário seleciona uma célula com o mouse.

```
procedure TForm1.StringGrid1Click(Sender: TObject);
begin
Label1.Caption:= IntToStr(StringGrid1.RowCpunt);
end;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBLookupListBox, TDecisionGrid, TIWDBGrid, TIWDBLookupListBox e TDBCtrlGrid
```

Durante a execução do aplicativo:

```
TDBLookupListBox, TDecisionGrid, TIWDBGrid, TIWDBLookupListBox e TDBCtrlGrid
```

RowHeights

Descrição

A propriedade RowHeights é declarada com uma array de inteiros, no qual cada item da array define a altura, em pixels, das células da linha especificada pelo índice. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe a altura das células da primeira linha de um componente StringGrid1 do tipo TStringGrid quando o usuário seleciona uma célula com o mouse.

```
procedure TForm1.StringGrid1Click(Sender: TObject);
begin
    Label1.Caption := IntToStr(StringGrid1.RowHeights[0]);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDrawGrid e TStringGrid
```

Rows

Descrição

A propriedade Rows é declarada como uma array de listas de strings, no qual cada lista armazena as strings das células de uma linha da grade. O índice da array define o número da linha a ser acessada, começando com 0. Na realidade, cada linha é tratada como uma lista de strings. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

A linha de código a seguir adiciona a string 'Nova string' à lista de strings correspondente à terceira linha de um componente chamado StringGrid1, do tipo TStringGrid:

```
StringGrid1.Rows[2].Add('Nova string');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TStringGrid
```

RowsAffected

Descrição

Esta propriedade é declarada como uma variável SQL e retorna o número de registros afetados por uma declaração SQL

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TADOCommand, TADODataset, TADOTable, TADOQuery, TADOStoredproc
```

Ruler

Descrição

Essa propriedade é declarada como uma variável do tipo TQRRuler e permite que se exiba, na fase de projeto, uma régua em um componente do tipo TQRBand para facilitar o posicionamento de outros componentes.

Tabela de Valores:

Valor	Significado
qrrNone	Nenhuma régua será exibida.
qrrInchesH	Régua horizontal graduada em polegadas.
qrrInchesV	Régua vertical graduada em polegadas.
qrrInchesHV	Réguas horizontal e vertical graduadas em polegadas.
qrrCmH	Régua horizontal graduada em centímetros.
qrrCmV	Régua vertical graduada em centímetros.
qrrCmHV	Réguas horizontal e vertical graduadas em centímetros.

Componentes aos quais se aplica:

Na fase de projeto:

```
TQRBand
```

Durante a execução do aplicativo:

```
TQRBand
```

Scaled

Descrição

A propriedade Scaled é declarada como uma variável booleana que determina se o formulário deve ser multiplicado por um fator de escala, de forma a que o valor definido na propriedade PixelsPerInch seja satisfeito.

Exemplo

Você pode definir o valor da propriedade Scaled de um formulário diretamente no Object Inspector ou mediante uma linha de código como:

```
Form1.Scaled := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TForm
```

ScrollBars

Descrição

A propriedade ScrollBars é declarada como uma variável do tipo TScrollStyle que define se o controle possui ou não barras de rolamento.

Tabela de Valores:

Valor	Significado
ssNone	Nenhuma barra de rolamento.
ssHorizontal	Barra de rolamento horizontal na base do controle.
ssVertical	Barra de rolamento vertical do lado direito do controle.
ssBoth	Barras de rolamento horizontal e vertical.

Exemplo

Você pode definir o valor da propriedade ScrollBars de um componente diretamente no Object Inspector ou mediante uma linha de código:

```
DBMemo1.ScrollBars:= ssBoth;
```

Componentes aos quais se aplica:

Na fase de projeto

```
TDBMemo, TDrawGrid, TMemo, TOutline, TRichEdit e TStringGrid
```

Durante a execução do aplicativo:

```
TDBMemo, TDrawGrid, TMemo, TOutline, TRichEdit e TStringGrid
```

ScrollPos

Descrição

Essa propriedade é declarada como uma variável inteira e armazena o valor corrente da propriedade Position. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir exibe o valor da propriedade ScrollPos de um objeto ControlScrollBar1, do tipo TControlScrollBar:

```
ShowMessage(IntToStr(ControlScrollBar1.ScrollPos);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TControlScrollBar
```

Sections

Descrição

Para controles do tipo THeader, a propriedade Sections é declarada como uma variável do tipo TStrings que associa, mediante uma lista de strings, um texto a cada uma das seções de um componente do tipo THeader.

Para controles do tipo THeaderControl, essa propriedade é usada na inserção de uma nova seção.

Exemplo

Você pode definir essa propriedade diretamente no Object Inspector com o String List Editor. Nesse caso, a primeira linha será o texto exibido pela a primeira seção (a da esquerda), a segunda linha será o texto exibido pela segunda seção (a segunda, da esquerda para a direita) e assim por diante.

O trecho de código a seguir altera o texto da primeira seção de um controle Header1 do tipo THeader quando o usuário dá um clique com o mouse sobre um formulário chamado Form1:

```
procedure TForm1.FormClick(Sender: TObject);
var
         NovaHints: TStringList;
begin
   NovaHints:= TStringList.Create;
   NovaHints.Add('Texto da primeira seção');
   Header1.Hints:= NovaHints;
end;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
THeader e THeaderControl
```

Durante a execução do aplicativo:

```
THeader e THeaderControl
```

SectionWidth

Descrição

A propriedade SectionWidth é declarada com uma array de inteiros, no qual cada item da array define a largura, em pixels, de uma seção de um componente do tipo THeader. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que um componente chamado Edit1 do tipo TEdit informe a largura da primeira seção de um componente Header1 do tipo THeader quando o usuário seleciona um botão chamado Button1 do tipo TButton.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
      Edit1.Text := IntToStr(Header1.SectionWidth[0]);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
THeader
```

SelCount

Descrição

A propriedade SelCount é uma variável inteira que especifica o número de itens selecionados no controle quando a sua propriedade MultiSelect é True. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel exiba o número de itens selecionados em um componente ListBox1 do tipo TListBox quando o usuário clicar com o mouse sobre Label1:

```
procedure TForm1.Label1Click(Sender: TObject);
begin
       Label1.Caption := IntToStr(ListBox1.SelCount);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBListBox, TDirectoryListBox, TFileListBox e TListBox
```

Selected

Descrição

A propriedade Selected é declarada como uma array de variáveis booleanas que informa se um item de um controle está selecionado. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe se o primeiro item em um componente ListBox1 do tipo TListBox está selecionado quando o usuário clicar com o mouse sobre Label1:

```
procedure TForm1.Label1Click(Sender: TObject);
begin
      if ListBox1.Selected[0] = True then Label1.Caption:= 'Primeiro Item Selecionado';
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBListBox, TDirectoryListBox, TFileListBox, TTreeView, TListView e TListBox
```

SelectedColor

Descrição

A propriedade SelectedColor é declarada como uma variável do tipo TColor e define a cor da guia selecionada em um componente do tipo TTabSet.

Exemplo

Você pode alterar a propriedade SelectedColor de um componente diretamente no Object Inspector ou por meio de uma linha de código, como:

```
TabSet1. SelectedColor := clRed;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTabSet
```

Durante a execução do aplicativo:

```
TTabSet
```

SelectedField

Descrição

A propriedade SelectedField é declarada como uma variável do tipo TField que indica o campo do banco de dados selecionado pelo componente. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Você pode usar um componente chamado Label1 do tipo TLabel para exibir o nome do campo selecionado em um componente DBGrid1 do tipo TDBGrid, com a seguinte linha de código:

```
Label1.Caption := DBGrid1.SelectedField.FieldName;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBGrid e TDBLookupList
```

SELECTEDINDEX

Descrição

A propriedade SelectedIndex é uma variável inteira que indica o índice do campo do banco de dados selecionado pelo componente. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode usar um componente chamado Label1 do tipo TLabel para exibir o índice do campo selecionado em um componente DBGrid1 do tipo TDBGrid, com a seguinte linha

```
de código:
Label1.Caption := IntToStr(DBGrid1.SelectedIndex);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBGrid e TDBLookupList
```

SELECTEDITEM

Descrição

A propriedade SelectedItem é uma variável do tipo inteiro longo (Longint) que indica o item que possui o foco em um componente do tipo TOutline. Essa propriedade só está disponível durante a execução de um aplicativo.

Exemplo

Você pode usar um componente chamado Label1 do tipo TLabel para exibir o número do item que possui o foco em um componente do tipo TOutline com a seguinte linha de código:

```
Label1.Caption := IntToStr(Outline1.SelectedItem);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TOutline
```

SELECTION

Descrição

A propriedade Selection é declarada como uma variável do tipo TGridRect que contém as coordenadas da(s) linha(s) e coluna(s) da(s) célula(s) selecionadas na grade.

Exemplo

O trecho de código a seguir seleciona as células contidas nas linhas 1 e 2 e nas colunas 3 e 4:

```
var
```

```
     SRect: TGridRect;
begin
     SRect.Top := 1;
     SRect.Left := 3;
     SRect.Bottom := 2;
     SRect.Right := 4;
     StringGrid1.Selection := SRect;
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

`TDBGrid e TStringGrid`

SelectSQL

Descrição

Essa propriedade armazena uma lista de strings contendo a declaração SQL usada para exibir o resultado de uma declaração SQL.

Componentes aos quais se aplica:

Na fase de projeto:

`TIBDataset, TIBUpdateSQL`

Durante a execução do aplicativo:

`TIBDataset, TIBUpdateSQL`

SelEnd

Descrição

Essa propriedade é declarada como uma variável inteira que define o valor final da faixa de valores do componente.

Exemplo

Você pode definir o valor dessa propriedade diretamente no Object Inspector ou mediante uma linha de código como:

```
TrackBar1.SelEnd:= 100;
```

Componentes aos quais se aplica:

Na fase de projeto:

`TTrackBar`

Durante a execução do aplicativo:

`TTrackBar`

SelLength

Descrição

A propriedade SelLength é uma variável inteira que retorna o comprimento (em caracteres) do texto selecionado no componente.

Exemplo

Você pode atribuir o valor da propriedade SelLength de um componente a uma variável inteira "a" com a seguinte linha de código:

```
a := Edit1.SelLength;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TComboBox, TDBComboBox, TDBEdit, TDBMemo, TDriveComboBox, TEdit, TFilterComboBox, TMaskEdit e TMemo
```

SelStart

Descrição

Para componentes dos tipos TComboBox, TDBComboBox, TDBEdit, TDBMemo, TDriveComboBox, TEdit, TFilterComboBox, TMaskEdit e TMemo, a propriedade SelStart é uma variável inteira que retorna a posição inicial da parte selecionada do texto de um controle, em que o primeiro caractere do texto do controle ocupa uma posição cujo valor é igual a 0. Essa propriedade só está disponível durante a execução do aplicativo.

Para componentes do tipo TTrackBar, essa propriedade é declarada como uma variável inteira que define o valor inicial da faixa de valores do componente.

Exemplo

Você pode atribuir o valor da propriedade SelStart do texto selecionado de um componente a uma variável inteira "a" com a seguinte linha de código:

```
a:= Edit1.SellStart;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTrackBar
```

Durante a execução do aplicativo:

```
TTrackBar, TComboBox, TDBComboBox, TDBEdit, TDBMemo, TDriveComboBox, TEdit, TFilterComboBox,
TMaskEdit e TMemo
```

SelText

Descrição

A propriedade SelText é uma variável do tipo string que contém a porção de texto do controle que se encontra selecionada. Você pode substituir o texto selecionado atribuindo uma string à variável SelText. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode atribuir o texto selecionado em um componente à propriedade Caption de um componente Label1 do tipo TLabel da seguinte forma:

```
procedure TForm1.Edit1MouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
     Label1.Caption := Edit1.SelText;
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TComboBox, TDBComboBox, TDBEdit, TDriveComboBox, TEdit, TFilterComboBox e TMaskEdit
```

Series

Descrição

Essa propriedade é declarada como uma array de objetos da classe TChartSeries e referencia todas as séries definidas para o gráfico exibido no componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart
```

Durante a execução do aplicativo:

```
TChart, TDBChart
```

SeriesColor

Descrição

A propriedade SelectedColor é declarada como uma variável do tipo TColor e define a cor default dos pontos da série representada pelo objeto.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChartSeries
```

Durante a execução do aplicativo:

```
TChartSeries
```

ServerConv

Descrição

A propriedade ServerConv é declarada como uma variável do tipo TDDEServerConv e define o componente do tipo TDDEServerConv ao qual esse item está associado em uma conversação DDE.

Exemplo

Você pode alterar o valor da propriedade ServerConv de um componente diretamente no Object Inspector ou por meio de uma linha de código, como:

```
DDEServerItem1. ServerConv := DDEServerConv1
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDDEServerConv
```

Durante a execução do aplicativo:

```
TDDEServerConv
```

ServiceApplication

Descrição

A propriedade ServiceApplication é declarada como uma variável do tipo string que define o nome do arquivo executável (sem a extensão .EXE) da aplicação servidora.

Exemplo

Você pode alterar o valor da propriedade ServiceApplication de um componente diretamente no Object Inspector ou por meio de uma linha de código, como:

```
DdeClientConv1. ServiceApplication := 'Programa';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDDEClientConv
```

Durante a execução do aplicativo:

```
TDDEClientConv
```

SHAPE

Descrição

A propriedade Shape é uma variável que define o formato geométrico de um componente. No caso de um componente do tipo TBevel, é uma variável do tipo TBevelShape, enquanto que no caso de um componente do tipo TShape, é do tipo TShapeType.

Tabela de Valores para componentes do tipo TBevel.

Nesse caso, a propriedade é uma variável do tipo TBevelShape.

Valor	Significado
bsBox	O chanfro assume a forma de uma caixa.
bsFrame	O chanfro assume a forma de um quadro.
bsTopLine	O chanfro se apresenta como uma linha no topo do controle.
bsBottomLine	O chanfro se apresenta como uma linha na base do controle.
bsLeftLine	O chanfro se apresenta como uma linha na lateral esquerda do controle.
bsRightLine	O chanfro se apresenta como uma linha na lateral direita do controle.

Tabela de Valores para componentes do tipo TShape.

Nesse caso, a propriedade é uma variável do tipo TShapeType.

Valor	Significado
stEllipse	O componente tem o formato de uma elipse.
stRectangle	O componente tem o formato de um retângulo.
stRoundRect	O componente tem o formato de um retângulo com cantos arredondados.
stRoundSquare	O componente tem o formato de um quadrado com cantos arredondados.
stSquare	O componente tem o formato de um quadrado.
stCircle	O componente tem o formato de um círculo.

Exemplo

Você pode alterar a propriedade Shape de um componente Bevel1 do tipo TBevel com a seguinte linha de código:

```
Bevel1.Shape := bsFrame;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBevel e TShape
```

Durante a execução do aplicativo:

```
TBevel e TShape
```

Sharable

Descrição

A propriedade Sharable é declarada como uma variável booleana que determina se mais de um aplicativo pode compartilhar um dispositivo multimídia.

Exemplo

Você pode definir o valor da propriedade Sharable diretamente no Object Inspector ou mediante uma linha de código como:

```
MediaPlayer1.Sharable := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMediaPlayer
```

Durante a execução do aplicativo:

```
TMediaPlayer
```

ShareImages

Descrição

Essa propriedade é declarada como uma variável do tipo booleana e indica se o controle destrói o seu handle quando a lista de imagens é destruída.

Componentes aos quais se aplica:

Na fase de projeto:

```
TImageList
```

Durante a execução do aplicativo:

```
TImageList
```

ShortCut

Descrição

A propriedade ShortCut consiste em uma seqüência de teclas que permite que um usuário acesse rapidamente um item de menu.

Para objetos da classe TAction, define a propriedade ShortCut dos itens de menu associados ao controle.

Exemplo

Você pode definir o valor da propriedade ShortCut diretamente no Object Inspector ou mediante uma linha de código como:

```
MenuItem1.ShortCut := ShortCut(Word('C'), [ssCtrl]);
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TAction, TMenuItem
```

Durante a execução do aplicativo:

```
TAction, TMenuItem
```

ShowAccelChar

Descrição

A propriedade ShowAccelChar é uma variável do tipo booleana que define se um caractere '&' deve ser utilizado para sublinhar a tecla aceleradora do componente ou ser realmente utilizado como o caractere '& '.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
Label1.ShowAccelChar := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TLabel
```

Durante a execução do aplicativo:

```
TLabel
```

ShowButtons

Descrição

Essa propriedade é declarada como uma variável do tipo booleana e indica se o controle deve exibir os sinais (+) e (-) à esquerda de cada item que possui uma lista de subitens. Esses botões, se exibidos, podem ser usados pelo usuário para exibir ou ocultar uma lista de subitens.

Componentes aos quais se aplica:

Na fase de projeto:

```
TTreeView
```

Durante a execução do aplicativo:

```
TTreeView
```

ShowColumnHeaders

Descrição

Essa propriedade é declarada como uma variável booleana e define se os cabeçalhos das colunas do controle devem ser exibidos.

Componentes aos quais se aplica:

Na fase de projeto:

```
TListView
```

Durante a execução do aplicativo:

```
TListView
```

ShowFocus

Descrição

Essa propriedade é declarada como uma variável booleana que define se um retângulo de foco deve ser desenhado no painel que exibe o registro corrente.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
DBCtrlGrid1.ShowFocus:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBCtrlGrid
```

Durante a execução do aplicativo:

```
TDBCtrlGrid
```

ShowHint

Descrição

A propriedade ShowHint é uma variável do tipo booleana que define se uma string de auxílio deve ou não ser exibida quando o usuário mantém o ponteiro do mouse sobre um controle. No caso de um componente do tipo TApplication, essa propriedade é definida apenas durante a execução do aplicativo; se o seu valor for False, nenhuma string de auxílio será exibida durante a execução do aplicativo, ainda que a propriedade ShowHint de todos os controles seja igual a True.

Exemplo

Coloque, em um formulário chamado Form1, um botão chamado Button1 e defina, no Object Inspector, sua propriedade Hint como Botão e ShowHint como True. Execute o aplicativo e a string de auxílio será exibida quando o mouse estiver sobre o botão. No arquivo de projeto, inclua a seguinte linha de código, antes de Application.Run:

```
Application.ShowHint := False;
```

Execute novamente o aplicativo. Nesse caso a string de auxílio não será exibida.

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles.

Durante a execução do aplicativo:

Todos os controles e os componentes do tipo TApplication.

Showing

Descrição

A propriedade Showing é uma variável do tipo booleana que especifica se um componente está ou não sendo exibido na tela. Se um componente-pai do componente atual tiver a sua propriedade Visible igual a False, então sua propriedade Showing terá o valor False. Essa propriedade só pode ser acessada durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Se um formulário estiver visível, você pode ocultá-lo com a seguinte linha de código:

```
Visible := not Showing;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

Todos os controles.

ShareInLegend

Descrição

Essa propriedade é declarada como uma variável do tipo booleana e define se a legenda da série deve ser exibida no componente Chart associado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChartSeries
```

Durante a execução do aplicativo:

```
TChartSeries
```

ShowLines

Descrição

Essa propriedade é declarada como uma variável booleana que define se o controle deve exibir linhas conectando itens ao seu componente-pai.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
TreeView1.ShowLines:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTreeView
```

Durante a execução do aplicativo:

```
TTreeView
```

ShowProgress

Descrição

Essa propriedade é declarada como uma variável booleana e define se uma caixa de diálogo com uma barra, que indica o progresso da tarefa de impressão do relatório, deve ser exibida.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
QuickReport1.ShowProgress:= True;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TQuickReport e TQRPrinter
```

Na fase de projeto:

```
TQuickReport e TQRPrinter
```

ShowRoot

Descrição

Essa propriedade é declarada como uma variável booleana que define se o controle deve exibir linhas conectando itens ao componente-raiz.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
TreeView1.ShowRoot:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTreeView
```

Durante a execução do aplicativo:

```
TTreeView
```

SimplePanel

Descrição

Essa propriedade é declarada como uma variável booleana e define se a barra de status exibe um único painel ou múltiplos painéis.

Componentes aos quais se aplica:

Na fase de projeto:

```
TStatusBar
```

Durante a execução do aplicativo:

```
TStatusBar
```

SimpleText

Descrição

Essa propriedade é declarada como uma variável do tipo string e define o texto exibido na barra de status quando o valor da propriedade SimplePanel é igual a True.

Componentes aos quais se aplica:

Na fase de projeto:

```
TStatusBar
```

Durante a execução do aplicativo:

```
TStatusBar
```

Size

Descrição

Para componentes do tipo TStringField, essa propriedade define o número de Bytes reservado para o campo.

Para componentes do tipo TBCDField, essa propriedade define o número de dígitos após o ponto decimal.

Para componentes dos tipos TAutoIncField, TFieldDef, TBlobField, TBytesField, TVarBytesField, TMemoField e TGraphicField, define o espaço que o campo ocupa quando armazenado em uma tabela.

Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBlobField, TBytesField, TFieldDef, TGraphicField, TIntegerField,
TMemoField, TStringField, TTimeField e TVarBytesField
```

SmallChange

Descrição

A propriedade SmallChange é declarada como uma variável do tipo TScrollBarInc que define quantas posições devem ser deslocadas na barra de rolagem quando o usuário dá um clique nas setas situadas nas extremidades da barra de rolagem ou pressiona as teclas de seta.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
ScrollBar1.SmallChange := 10
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TScrollBar
```

Durante a execução do aplicativo:

```
TScrollBar
```

SmallImages

Descrição

Essa propriedade define as imagens exibidas pelo componente quando a sua propriedade ViewStyle possui o valor vsSmallIcon.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TListView
```

Sorted

Descrição

A propriedade Sorted é uma variável booleana que define se os itens de um componente dos tipos TComboBox, TIWComboBox, TIWListBox ou TListBox estarão alfabeticamente ordenados. Se quiser

ordenar alfabeticamente os itens do controle, basta atribuir o valor True à sua propriedade Sorted. Quando você adiciona ou insere itens a um controle cuja propriedade Sorted é True, o Delphi os coloca automaticamente na posição correta.

Exemplo

Você pode alterar a propriedade Sorted de um componente diretamente no Object Inspector ou incluindo uma linha de código como:

```
ListBox1Sorted:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TComboBox, TDBComboBox, TDBListBox, TIWComboBox, TIWDBComboBox, TIWDBListBox e TListBox
```

Durante a execução do aplicativo:

```
TComboBox, TDBComboBox, TDBListBox, TIWComboBox, TIWDBComboBox, TIWDBListBox, TListBox e TStringList
```

SortType

Descrição

A propriedade SortType é declarada como uma variável do tipo TSortType que define quando e como uma lista de itens deve ser ordenada.

Tabela de Valores:

Valor	Significado
nsNone	Nenhuma reordenação.
nsData	Os itens são reordenados quando um objeto de dados é alterado.
nsText	Os itens são reordenados quando um rótulo é alterado.
nsBoth	Os itens são reordenados quando um objeto de dados ou um rótulo é alterado.

Exemplo

Você pode definir o valor da propriedade ScrollBars de um componente diretamente no Object Inspector ou mediante uma linha de código:

```
TreeView1.SortType:= nsBoth;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTreeView e TListView
```

Durante a execução do aplicativo:

```
TTreeView e TListView
```

Source

Descrição

Para componentes do tipo TBatchMove, a propriedade Source é declarada como uma variável do tipo TDatasSet (TTable ou TQuery, TDecisionQuery) que especifica o componente ao qual está associada a tabela-origem em uma operação realizada por um componente do tipo TBatchMove.

Para componentes do tipo EOLEException, a propriedade Source é declarada como uma variável do tipo string e retorna o nome da aplicação que gerou a exceção.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou por meio de uma linha de código, como:

```
BatchMove1.Source := Table1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBatchMove
```

Durante a execução do aplicativo:

```
EoleException, TBatchMove
```

Spacing

Descrição

A propriedade Spacing é declarada como uma variável inteira que define a distância, em pixels, entre a imagem gráfica (definida na propriedade Glyph) e o texto (definido na propriedade Caption). Se for igual a 0, não haverá espaço, e se for igual a -1, o texto aparecerá centralizado no espaço entre a imagem e a extremidade do botão.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
BitBtn1.Spacing := 4
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn1 e TSpeedButton
```

Durante a execução do aplicativo:

```
TBitBtn1 e TSpeedButton
```

SparseCols

Descrição

Essa propriedade é declarada como uma variável booleana e define se as colunas em branco devem ser removidas da projeção corrente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDecisionSource
```

Durante a execução do aplicativo:

```
TDecisionSource
```

SparseRows

Descrição

Essa propriedade é declarada como uma variável booleana e define se as linhas em branco devem ser removidas da projeção corrente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDecisionSource
```

Durante a execução do aplicativo:

```
TDecisionSource
```

SQL

Descrição

Para componentes dos tipos TIBSQL, TQuery, TDecisionQuery, a propriedade SQL é declarada como uma variável do tipo TStrings que armazena a declaração ou comando SQL a ser executado quando os métodos Open e SQL forem chamados.

Para componentes dos tipos TUpdateSQL e TIBUpdateSQL, essa propriedade retorna à declaração SQL cujo índice é igual a UpdateKind. UpdateKind pode ser igual a ukModify, ukDelete ou ukInsert. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBSQL, TADOQuery, TQuery, TDecisionQuery
```

Durante a execução do aplicativo:

```
TIBSQL, TADOQuery, TQuery, TdecisionQuery, TIBUpdateSQL e TUpdateSQL
```

SQLCompatible

Descrição

Essa propriedade é declarada como uma variável booleana e deve ser usada quando uma mensagem de erro do Borland Database Engine indicar que o banco de dados utilizado não suporta a função RecordCount.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
QuickReport1.SQLCompatible:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TQuickReport
```

Durante a execução do aplicativo:

```
TQuickReport
```

SQLConnection

Descrição

Essa propriedade é definida como um objeto da classe TSQLConnection, e define o nome do componente SQLConnection através do qual será feita a conexão ao servidor do banco de dados.

Componentes aos quais se aplica:

Na fase de projeto:

```
TSimpleDataset, TSQLDataset, TSQLTable, TSQLQuery e TSQLStoredproc
```

Durante a execução do aplicativo:

```
TSimpleDataset, TSQLDataset, TSQLTable, TSQLQuery e TSQLStoredproc
```

SQLDialect

Descrição

Esta propriedade é declarada como uma variável do tipo inteiro e define o tipo de dialeto SQL usado pela aplicação cliente que acessa o banco de dados.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDataBase
```

Durante a execução do aplicativo:

```
TIBDataBase
```

SQLObjectCount

Descrição

Esta propriedade é declarada como uma variável do tipo inteiro que define o número de objetos de acesso via SQL associados ao componente que representa o banco de dados e que se encontram ativos.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDatabase, TIBTransaction
```

Durante a execução do aplicativo:

```
TIBDatabase, TIBTransaction
```

SQLObjects

Descrição

Esta propriedade é declarada como uma variável do tipo inteiro que define o número de objetos SQL do banco de dados.

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBDatabase
```

Durante a execução do aplicativo:

```
TIBDatabase
```

Start

Descrição

A propriedade Start é declarada como uma variável do tipo inteiro longo (Longint) que define a posição onde será iniciada a execução ou gravação em um dispositivo multimídia. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir exibe uma mensagem com o valor da propriedade Start durante a execução de um dispositivo multimídia:

```
ShowMessage(IntToStr(MediaPlayer1.Start));
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

StartFrame

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro e define o primeiro quadro a ser executado pelo clipe de vídeo exibido pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAnimate
```

StartMargin

Descrição

A propriedade StartMargin é declarada como uma variável inteira e determina a distância, em pixels, da guia mais à esquerda entre as visíveis no controle e a extremidade esquerda do controle.

Exemplo

O valor da propriedade StartMargin pode ser alterado diretamente no Object Inspector ou mediante a inclusão de uma linha de código, como:

```
TabSet1. StartMargin := 6;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTabSet
```

Durante a execução do aplicativo:

```
TTabSet
```

StartPos

Descrição

A propriedade StartPos é declarada como uma variável do tipo inteiro longo (Longint) que define a posição inicial para execução ou gravação em um dispositivo multimídia. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir exibe uma mensagem com o valor da propriedade StartPos durante a execução de um dispositivo multimídia:

```
ShowMessage(IntToStr(MediaPlayer1.StartPos));
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

State

Descrição

Para componentes dos tipos TCheckBox e TDBCheckBox, a propriedade State é uma variável do tipo TCheckBoxState que define os vários estados que podem ser assumidos pelo componente.

Para componentes dos tipos TDataSource, TTable, TQuery, TDecisionQuery e TStoredProc, a propriedade State é uma variável do tipo TDataSetState que define os estados que podem ser assumidos por um banco de dados associado. Para esses componentes, essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Tabela de Valores para componentes dos Tipos TCheckBox e TDVCheckBox.

Valor	Significado
cbUnchecked	O componente não apresenta um marca de verificação, o que indica que o componente não está selecionado.
cbChecked	O componente apresenta um marca de verificação na cor preta, o que indica que o componente está selecionado.
cbGrayed	O componente apresenta um marca de verificação na cor cinza, o que indica que o componente está em um terceiro estado. Esse terceiro estado só é possível se o componente tiver a sua propriedade AllowGrayed igual a True.

Tabela de Valores para componentes dos tipos TDataSource, TTable, TQuery, TDecisionQuery e TStoredProc:

Valor	Significado
dsInactive	O banco de dados está fechado.
dsBrowse	O banco de dados está fechado.
dsEdit	O banco de dados está sendo editado.
dsInsert	O banco de dados permite inserção de registros.
dsSetKey	O banco de dados está definindo registros-chave.
dsCalcFields	O evento OnCalcFields foi acionado.

Exemplo

Você pode definir o valor da propriedade State de um componente diretamente no Object Inspector ou mediante uma linha de código:

```
CheckBox1.State:= cbChecked;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TCheckBox TDBCheckBox, TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataset,
TDecisionQuery, TIBDataset, TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery,
TDecisionQuery e TStoredProc
```

Durante a execução do aplicativo:

```
TADODataset, TADOQuery, TADOStoredProc, TADOTable, TClientDataset, TDecisionQuery, TIBDataset,
TIBQuery, TIBStoredProc, TIBTable, TIBTransaction, TTable, TQuery, TDecisionQuery e TStoredProc
```

STATEIMAGES

Descrição

Essa propriedade define a imagem exibida à esquerda do ícone de um item.

Componentes aos quais se aplica:

Na fase de projeto:

```
TListView e TTreeView
```

Durante a execução do aplicativo:

```
TListView e TTreeView
```

STATUS

Descrição

A propriedade Status é declarada como uma variável do tipo TQRPrinterStatus que define o estado do objeto que representa a impressora. Essa propriedade só está disponível durante a execução do aplicativo.

Tabela de Valores:

Valor	Significado
mpReady	Componente pronto para iniciar um novo trabalho de impressão.
mpBusy	Componente ocupado com um trabalho de impressão.
mpFinished	Trabalho de impressão finalizado.
mpPrinting	Trabalho de impressão sendo enviado para a impressora.
mpPreviewing	Pré-visualização de impressão ativa.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDRPrinter
```

Step

Descrição

A propriedade Step é uma variável inteira que define o incremento usado na variação do valor da propriedade Position do componente.

Exemplo

Você pode alterar o valor da propriedade Step diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
ProgressBar1.Step:= 10;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TProgressBar
```

Durante a execução do aplicativo:

```
TProgressBar
```

StmtHandle

Descrição

A propriedade StmtHandle é declarada como uma variável do tipo HDBIStmt que permite acesso direto às funções da API do Borland Database Engine (BDE). Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TQuery, TDecisionQuery e TStoredProc
```

StopFrame

Descrição

Essa propriedade é declarada como uma variável do tipo inteiro e define o último quadro a ser executado pelo clipe de vídeo exibido pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAnimate
```

Storage

Descrição

A propriedade Storage é declarada como uma variável do tipo TStorage que permite o acesso à interface IStorage de um componente do tipo TOLEContainer. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

Você pode exibir o valor da propriedade Storage de um componente do tipo TOLEContainer mediante uma linha de código como:

```
ShowMessage(OLEContainer1.Storage);
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOLEContainer
```

Durante a execução do aplicativo:

```
TOLEContainer
```

StoredDefs

Descrição

A propriedade StoredDefs é declarada como uma variável booleana que determina se as definições de campos e índices devem ser armazenadas de forma persistente no arquivo DFM que armazena a definição do formulário.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOTable, TIBTable, TTable, TClientDataset
```

Durante a execução do aplicativo:

```
TADOTable, TIBTable, TTable, TClientDataset
```

StoredProcName

Descrição

Essa propriedade é declarada como uma variável do tipo string que define o nome do procedimento a ser executado em um servidor.

Exemplo

Você pode alterar o valor da propriedade StoredProcName diretamente no Object Inspector ou mediante uma linha de código como:

```
StoredProc1. StoredProcName := 'Exemplo';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TIBStoredproc, TStoredProc
```

Durante a execução do aplicativo:

```
TIBStoredproc, TStoredProc
```

Stretch

Descrição

A propriedade Stretch é uma variável booleana que define se uma imagem na forma de Bitmap ou Metafile deve ser redimensionada de forma a assumir o tamanho e a forma do controle que irá exibi-la.

Exemplo

Você pode definir o valor da propriedade Stretch de um componente diretamente no Object Inspector ou mediante uma linha de código:

```
Image1.Stretch := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TImage e TDBImage
```

Durante a execução do aplicativo:

```
TImage e TDBImage
```

STRINGS

Descrição

Essa variável é declarada como um array de strings e permite o acesso a uma string de uma lista de strings. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TStringList e TStrings
```

STYLE

Descrição

A propriedade Style está associada ao estilo de apresentação de um controle ou componente.

Tabela de Valores para um componente do tipo TBevel:

Nesse caso, a propriedade define se o chanfro será apresentado em relevo ou como uma depressão.

Valor	Significado
bsLowered	O chanfro é representado como uma depressão.
bsRaised	O chanfro é representado em relevo.

Tabela de Valores para um objeto do tipo TPen.

Nesse caso, a propriedade é uma variável do tipo TPenStyle que define o estilo de desenho da caneta.

Valor	Significado
psSolid	A caneta desenha uma linha sólida.
psDash	A caneta desenha uma linha tracejada.
psDot	A caneta desenha uma linha pontilhada.
psDashDot	A caneta desenha uma linha no estilo traço-ponto.
psDashDotDot	A caneta desenha uma linha no estilo traço-dois pontos.
psClear	A caneta desenha uma linha invisível.
psInsideFrame	A caneta desenha uma linha interna ao quadro de figuras fechadas que possuem um retângulo circunscrito.

Tabela de Valores para um objeto do tipo TBrush:

Nesse caso, a propriedade é uma variável do tipo TBrushStyle que define o estilo de preenchimento de um pincel para janelas e formas gráficas.

Valor	Significado
bsSolid	Preenchimento total (sólido).
bsClear	Transparente.
bsBDiagonal	Hachura em diagonal com a seguinte inclinação: ///.
bsFDiagonal	Hachura em diagonal com a seguinte inclinação: \\\.
bsCross	Hachuras ortogonais cruzadas.
bsDiagCross	Hachuras diagonais cruzadas.
bsHorizontal	Hachuras horizontais.
bsVertical	Hachuras verticais.

Tabela de Valores para um objeto do tipo TFont:

Nesse caso, a propriedade é um conjunto de variáveis do tipo TFontStyles que define o estilo da fonte.

Valor	Significado
fsBold	A fonte está em negrito.
fsItalic	A fonte está em itálico.
fsUnderline	A fonte está sublinhada.
fsStrikeout	A fonte é cortada por uma linha horizontal.

Tabela de Valores para componentes dos tipos TComboBox e FDBCombo:

Nesse caso, a propriedade é uma variável do tipo TComboBoxStyle que define o estilo de exibição dos itens do componente.

Valor	Significado
csDropDown	Cria uma lista drop-down com uma caixa de edição na qual o usuário pode digitar texto.
csSimple	Cria uma caixa de edição sem uma lista drop-down.
csDropDownList	Cria uma lista drop-down sem uma caixa de edição.
csOwnerDrawFixed	Cria uma lista drop-down sem caixa de edição em que os itens podem ser qualquer objeto definido pelo usuário e não apenas strings, sendo que todos têm altura fixa.
csOwnerDrawVariable	Cria uma lista drop-down sem caixa de edição em que os itens podem ser qualquer objeto definido pelo usuário e não apenas strings, sendo que não têm altura fixa.

Tabela de Valores para um componente do tipo TListBox e TDBListBox:

Nesse caso, a propriedade é uma variável do tipo TListBoxStyle que define o estilo de exibição dos itens do componente.

Valor	Significado
lbStandard	Todos os itens são strings e possuem a mesma altura.
lbOwnerDrawFixed	Os itens podem ser qualquer objeto definido pelo usuário e não apenas strings, sendo que todos têm altura fixa, definida pela propriedade ItemHeight.
lbOwnerDrawVariable	Os itens podem ser qualquer objeto definido pelo usuário e não apenas strings, sendo que não têm altura fixa.

Tabela de Valores para um componente dos tipos TBitBtn e TSpeedButton:

Nesse caso, a propriedade é uma variável do tipo TButtonStyle que define o aspecto de um botão com bitmap.

Valor	Significado
bsAutoDetect	O bitmap possui o aspecto da versão do Windows sob a qual o aplicativo está sendo executado.
bsWin31	O bitmap possui o aspecto da versão 3.1 do Windows, independentemente da versão do Windows sob a qual o aplicativo está sendo executado.
bsNew	O bitmap possui um novo aspecto, independentemente da versão do Windows sob a qual o aplicativo está sendo executado.

Tabela de Valores para um componente do tipo TTabSet:

Nesse caso, a propriedade é uma variável do tipo TTabStyle que define o aspecto do componente.

Valor	Significado
tsStandard	Cada componente tab do conjunto TabSet tem o tamanho e a aparência default.
tsOwnerDraw	Cada componente tab do conjunto TabSet tem a altura definida pela propriedade TabHeight e a largura necessária para conter o texto ou glyph. Nesse caso, o componente tab pode exibir objetos que não sejam strings.

Tabela de Valores para um componente do tipo TOutline:

Nesse caso, a propriedade é uma variável do tipo TOutline que define como o componente exibe seus itens.

Valor	Significado
otStandard	Os itens são desenhados de acordo com o valor especificado na propriedade OutlineStyle.
otOwnerDraw	Os itens são desenhados no Canvas de acordo com o código escrito no programa.

Tabela de Valores para um componente do tipo TDBLookupCombo:

Nesse caso, a propriedade é uma variável do tipo TDBLookupComboStyle que define como o componente exibe seus itens.

Valor	Significado
csDropDown	Cria uma lista drop-down com uma caixa de edição na qual o usuário pode digitar texto.
csDropDownList	Cria uma lista drop-down sem uma caixa de edição.

Exemplo

Você pode alterar a propriedade Style de um componente Bevel1 do tipo TBevel com a seguinte linha de código:

```
Bevel1.Style := bsRaised;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TFont, TPen, TBrush, TBevel, TBitBtn, TComboBox, TDBComboBox, TDBListBox, TDBLookupCombo,
TListBox, TOutline e TTabSet
```

Durante a execução do aplicativo:

```
TFont, TPen ; TBrush, TBevel, TBitBtn, TComboBox, TDBComboBox, TDBListBox, TDBLookupCombo,
TListBox, TOutline e TTabSet
```

SUSPENDED

Descrição

Essa variável é declarada como uma variável booleana que define se a execução da Thread representada pelo objeto foi suspensa.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TThread
```

TABHEIGHT

Descrição

A propriedade TabHeight é declarada como uma variável inteira que define a altura, em pixels, das guias de um controle.

Componentes aos quais se aplica:

Na fase de projeto:

```
TPageControl, TTabSet e TTabControl
```

Durante a execução do aplicativo:

```
TPageControl, TTabSet e TTabControl
```

TabIndex

Descrição

A propriedade TabIndex é declarada como uma variável inteira que define a guia correntemente selecionada.

Componentes aos quais se aplica:

Na fase de projeto:

```
TPageControl, TTabSet e TTabControl
```

Durante a execução do aplicativo:

```
TPageControl, TTabSet e TTabControl
```

Table

Descrição

Essa propriedade retorna o nome do componente Table ao qual o relatório está vinculado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TRvTableConnection
```

Durante a execução do aplicativo:

```
TRvTableConnection
```

TableDirect

Descrição

Essa propriedade é definida como uma variável booleana, e define se o acesso à tabela é feito de forma direta, ou se é necessária a criação de uma declaração SQL em background.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOTable
```

Durante a execução do aplicativo:

```
TADOTable
```

TableLevel

Descrição

A propriedade TableLevel é declarada como uma variável inteira que define o driver do BDE vinculado à tabela representada pelo componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TTable
```

TableName

Descrição

Essa propriedade é declarada como uma variável do tipo TFileName e especifica a tabela a que o componente estará ligado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TADOTable, TIBTable, TTable
```

Durante a execução do aplicativo:

```
TADOTable, TIBTable, TTable
```

TABLETYPE

Descrição

Essa propriedade é declarada como uma variável do tipo TTableType e especifica o tipo da tabela a que o componente estará ligado (não vale para tabelas SQL).

Componentes aos quais se aplica:

Na fase de projeto:

```
TTable
```

Durante a execução do aplicativo:

```
TTable
```

TABORDER

Descrição

A propriedade TabOrder é uma variável do tipo TTabOrder que define a ordem segundo a qual os diversos controles de um formulário recebem o foco da aplicação quando o usuário pressiona a tecla Tab. Se você altera a propriedade TabOrder de qualquer componente, o Delphi altera automaticamente o valor da propriedade TabOrder dos demais componentes, de forma que não haja duplicação do valor da propriedade.

Exemplo

Você pode definir o valor da propriedade TabOrder de um componente diretamente no Object Inspector ou mediante uma linha de código:

```
BitBtn1.TabOrder:= 5;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn, TDirectoryListBox, TPageControl, TButton, TDrawGrid, TPanel, TCheckBox,
TDriveComboBox, TProgressBar, TComboBox, TEdit, TRadioButton, TDBCheckBox, TFileListBox,
TRadioGroup, TDBComboBox, TFilterComboBox, TRichEdit, TDBCtrlGrid, TScrollBar, TDBCtrlPanel,
TGroupBox, TScrollBox, TDBEdit, THeader, TStatusBar, TDBGrid, THeaderControl, TStringGrid,
TDBImage, THotKey, TTabbedNotebook, TDBListBox, TListBox, TTabControl, TDBLookupCombo,
TListView, TTabSet, TDBLookupComboBox, TMaskEdit, TTabSheet, TDBLookupList, TMediaPlayer,
TTrackBar, TDBLookupListBox, TMemo, TTreeView, TDBMemo, TNotebook, TUpDown, TDBNavigator,
TOLEContainer, TDBRadioGroup e TOutline
```

Durante a execução do aplicativo:

```
TBitBtn, TDirectoryListBox, TPageControl, TButton, TDrawGrid, TPanel, TCheckBox,
TDriveComboBox, TProgressBar, TComboBox, TEdit, TRadioButton, TDBCheckBox, TFileListBox,
TRadioGroup, TDBComboBox, TFilterComboBox, TRichEdit, TDBCtrlGrid, TScrollBar, TDBCtrlPanel,
TGroupBox, TScrollBox, TDBEdit, THeader, TStatusBar, TDBGrid, THeaderControl, TStringGrid,
TDBImage, THotKey, TTabbedNotebook, TDBListBox, TListBox, TTabControl, TDBLookupCombo,
TListView, TTabSet, TDBLookupComboBox, TMaskEdit, TTabSheet, TDBLookupList, TMediaPlayer,
TTrackBar, TDBLookupListBox, TMemo, TTreeView, TDBMemo, TNotebook, TUpDown, TDBNavigator,
TOLEContainer, TDBRadioGroup e TOutline
```

Tabs

Descrição

A propriedade Tabs é declarada como uma lista de strings que armazena o texto exibido para as diversas guias do controle.

Exemplo

Você pode definir o valor da propriedade Tabs de um controle diretamente no Object Inspector, usando o String List Editor, ou mediante uma linha de código como:

```
TabSet1.Tabs.Add('Nova guia');
```

Essa linha de código, além de acrescentar uma string à lista definida na propriedade Tabs, cria uma nova guia que exibirá o texto da string.

Componentes aos quais se aplica:

Na fase de projeto:

```
TTabControl e TTabSet
```

Durante a execução do aplicativo:

```
TTabControl e TTabSet
```

TabsPerRow

Descrição

A propriedade TabsPerRow é uma variável inteira que define o número de guias que aparecem em cada linha no topo do controle.

Exemplo

Você pode definir o valor da propriedade TabsPerRow de um controle diretamente no Object Inspector ou mediante uma linha de código como:

```
TabbedNotebook1.TabsPerRow := 2;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTabbedNotebook
```

Durante a execução do aplicativo:

```
TTabbedNotebook
```

TabStop

Descrição

A propriedade TabStop é uma variável do tipo booleana que define se um controle pode ou não receber o foco da aplicação quando o usuário pressiona a tecla Tab.

Exemplo

Você pode definir o valor da propriedade TabStop de um controle diretamente no Object Inspector ou mediante uma linha de código:

```
BitBtn1.TabStop:= False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TBitBtn, TDirectoryListBox, TPageControl, TButton, TDrawGrid, TPanel, TCheckBox, TDriveComboBox,
TProgressBar, TComboBox, TEdit, TRadioButton, TDBCheckBox, TFileListBox, TRadioGroup, TDBComboBox,
TFilterComboBox, TRichEdit, TDBCtrlGrid, TScrollBar, TDBCtrlPanel, TGroupBox, TScrollBox, TDBEdit,
THeader, TStatusBar, TDBGrid, THeaderControl, TStringGrid, TDBImage, THotKey, TTabbedNotebook,
TDBListBox, TListBox, TTabControl, TDBLookupCombo, TListView, TTabSet, TDBLookupComboBox,
TMaskEdit, TTabSheet, TDBLookupList, TMediaPlayer, TTrackBar, TDBLookupListBox, TMemo, TTreeView,
TDBMemo, TNotebook, TUpDown, TDBNavigator, TOLEContainer, TDBRadioGroup e TOutline
```

Durante a execução do aplicativo:

```
TBitBtn, TDirectoryListBox, TPageControl, TButton, TDrawGrid, TPanel, TCheckBox, TDriveComboBox,
TProgressBar, TComboBox, TEdit, TRadioButton, TDBCheckBox, TFileListBox, TRadioGroup, TDBComboBox,
TFilterComboBox, TRichEdit, TDBCtrlGrid, TScrollBar, TDBCtrlPanel, TGroupBox, TScrollBox, TDBEdit,
THeader, TStatusBar, TDBGrid, THeaderControl, TStringGrid, TDBImage, THotKey, TTabbedNotebook,
TDBListBox, TListBox, TTabControl, TDBLookupCombo, TListView, TTabSet, TDBLookupComboBox,
TMaskEdit, TTabSheet, TDBLookupList, TMediaPlayer, TTrackBar, TDBLookupListBox, TMemo, TTreeView,
TDBMemo, TNotebook, TUpDown, TDBNavigator, TOLEContainer, TDBRadioGroup e TOutline
```

TABSTOPS

Descrição

A propriedade TabStops é declarada como uma array de variáveis booleanas que define as colunas para as quais o usuário pode se mover usando a tecla Tab ou Shift+Tab. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Se você não quiser que o usuário se desloque para a segunda coluna de uma grade usando Tab ou Shift+Tab, inclua a seguinte linha de código:

```
StringGrid1.TabStops[1] := False;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDBGrid e TStringGrid
```

TABWIDTH

Descrição

A propriedade TabWidth é declarada como uma variável inteira que define a largura, em pixels, das guias de um controle.

Componentes aos quais se aplica:

Na fase de projeto:

```
TPageControl e TTabControl
```

Durante a execução do aplicativo:

```
TPageControl e TTabControl
```

TAG

Descrição

A propriedade Tag é uma variável do tipo Longint que o Delphi coloca à disposição do usuário, que pode atribuir o significado mais conveniente.

Exemplo

Você pode alterar a propriedade Tag de um componente Button1 do tipo TButton com a seguinte linha de código:

```
Button1.Tag := variável_inteira;
```

Componentes aos quais se aplica:

Na fase de projeto:

Todos os componentes.

Durante a execução do aplicativo:

Todos os componentes.

Temporary

Descrição

A propriedade Temporary é declarada como uma variável booleana e determina se o componente foi criado apenas porque não havia um componente do tipo TDatabase disponível quando uma tabela foi aberta. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDatabase
```

Terminated

Descrição

A propriedade Terminated é uma variável do tipo booleana que especifica se a aplicação recebeu do Windows a mensagem WM_QUIT para que seja encerrada, o que ocorre normalmente quando se fecha o formulário principal. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Para objetos da classe TThread, indica se sua execução deve ser finalizada.

Exemplo

O trecho de código a seguir exibe uma mensagem quando a aplicação estiver para ser encerrada:

```
if Application.Terminated = True then ShowMessage('A aplicação será encerrada');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TApplication
```

Text

Descrição

A propriedade Text é uma variável do tipo string cujo significado depende do tipo de componente a que se refere:

Para componentes dos tipos TComboBox, TDBComboBox, TIWComboBox e TIWDBComboBox, a propriedade Text define o primeiro item que aparece na lista de itens do componente quando a aplicação é executada.

Para componentes do tipo TDriveComboBox, a propriedade Text é igual à propriedade Drive.

Para componentes do tipo TFilterComboBox, a propriedade Text define o primeiro filtro que aparece na lista de filtros do componente quando a aplicação é executada.

Para componentes dos tipos TDBLookupCombo, TDBLookupComboBox, TIWDBLookupCombo, TIWDBLookupComboBox, a propriedade Text define o valor do campo do registro corrente.

Para componentes dos tipos TDBEdit, TDBMemo, TEdit, TIWDBEdit, TIWDBMemo, TIWEdit, TIWMemo, TMaskEdit e TMemo, essa propriedade é declarada como uma variável TCaption que define o texto exibido pelo componente, com um máximo de 255 caracteres.

Para qualquer um dos componentes citados, exceto TEdit, TMaskEdit e TComboBox com a propriedade Style igual a csDropDown, essa propriedade só está disponível durante a execução do aplicativo.

Para componentes dos tipos TDDEClientItem e TDDEServerItem, essa propriedade define o texto a ser transferido em uma conversação DDE.

Para objetos do tipo TOutlineNode, é declarada como uma variável do tipo string e armazena o texto que identifica o item.

Para componentes dos tipos TAutoIncField, TBCDField, TBlobField, TBytesField, TFieldDef, TGraphicField, TIntegerField, TMemoField, TStringField, TTimeField e TVarBytesField, essa propriedade é declarada como uma variável do tipo string que armazena como uma string o valor da propriedade.

Exemplo

Você pode alterar a propriedade Text de um componente ComboBox1 do tipo TComboBox com a propriedade Style igual a csDropDown diretamente no Object Inspector ou com a seguinte linha de código:

```
ComboBox1.Text:= 'Texto';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDDEClientItem, TDDEServerItem, TEdit, TIWEdit, TMaskEdit e TComboBox com a propriedade Style
igual a csDropDown
```

Durante a execução do aplicativo:

```
TAutoIncField, TOutlineNode, TParam, TTreeNode, TStrings, TStringList, THeaderSection,
TDBComboBox, TDBEdit, TDBMemo, TDDEClientItem, TDDEServerItem, TDriveComboBox, TEdit,
TFilterComboBox, TIWDBComboBox, TIWDBEdit, TIWEdit, TIWDBMemo, TMaskEdit, TMemo, TQuery,
TDecisionQuery, TBCDField, TBooleanField, TCurrencyField, TDateField, TDateTimeField,
TFloatField, TIntegerField, TSmallintField, TStringField, TTimeField, TWordField e TComboBox
com a propriedade Style igual a csDropDown
```

TextAlign

Descrição

Esta propriedade define como um texto será alinhado verticalmente em um Canvas, podendso assumir o valor taTop ou taBottom

Durante a execução do aplicativo:

```
TCanvas
```

TEXTCASE

Descrição

A propriedade TextCase é declarada como uma variável do tipo TTextCase que define se o nome exibido pela propriedade Text deve aparecer em letras maiúsculas ou minúsculas.

Tabela de Valores:

Valor	Significado
tcLowerCase	O nome exibido pela propriedade Text é apresentado em letras minúsculas.
tcUpperCase	O nome exibido pela propriedade Text é apresentado em letras maiúsculas.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou por meio de uma linha de código como:

```
DriveComboBox1.TextCase := tcUpperCase;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDriveComboBox
```

Durante a execução do aplicativo:

```
TDriveComboBox
```

THREADID

Descrição

Essa variável é declarada como uma variável do tipo THandle que identifica a Thread com o sistema operacional e permite que esta possa ser passada como parâmetros em funções da API do Windows.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TThread
```

TICKMARKS

Descrição

A propriedade TickMarks é declarada como uma variável do tipo TTickMark que define o tipo de marca usada para graduar os valores exibidos no componente.

Tabela de Valores:

Valor	Significado
tmBottomRight	As marcas são exibidas abaixo ou à direita do componente.
tmTopLeft	As marcas são exibidas acima ou à esquerda do componente.
tmBoth	As marcas são exibidas dos dois lados do componente.

Exemplo

Você pode definir o valor da propriedade TickMarks de um componente diretamente no Object Inspector ou mediante uma linha de código:

```
TrackBar1.TickMarks:= tmBoth;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTrackBar
```

Durante a execução do aplicativo:

```
TTrackBar
```

TICKSTYLE

Descrição

A propriedade TickStyle é declarada como uma variável do tipo TTickStyle que define o estilo de marca usada para graduar os valores exibidos no componente.

Tabela de Valores:

Valor	Significado
tsAuto	As marcas são exibidas automaticamente a cada incremento de valor. A propriedade Frequency só pode ser usada quando esse é o estilo definido para o componente.
tsManual	As marcas são exibidas nas extremidades do componente e nas posições definidas pelo método SetTick.
tsNone	O componente não exibe nenhuma marca.

Exemplo

Você pode definir o valor da propriedade TickStyle de um componente diretamente no Object Inspector ou mediante uma linha de código:

```
TrackBar1.TickStyle:= tsAuto;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTrackBar
```

Durante a execução do aplicativo:

```
TTrackBar
```

TILEMODE

Descrição

Essa propriedade é declarada como uma variável do tipo TTileMode e determina se as janelas-filhas de uma aplicação MDI devem ser automaticamente reposicionadas e redimensionadas quando a aplicação chama o método Tile. Essa propriedade só está disponível durante a execução do aplicativo.

Tabela de Valores:

Valor	Significado
tbHorizontal	Cada formulário terá suas dimensões alteradas de forma a preencher toda a largura da janela-pai.
tbVertical	Cada formulário terá suas dimensões alteradas de forma a preencher toda a altura da janela-pai.

Exemplo

Você pode alterar o valor da propriedade TileMode mediante a inclusão de uma linha de código, como:

```
Form1.TileMode := tbHorizontal;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TForm
```

TimeFormat

Descrição

A propriedade TimeFormat é declarada como uma variável do tipo TMPTimeFormats e define o formato usado para interpretar informações de posição em propriedades como StartPos, Length, Position, Start e EndPos em um dispositivo multimídia. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar o valor da propriedade TimeFormat mediante a inclusão de uma linha de código como:

```
TimeFormat := tfHMS;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

Timers

Descrição

Essa propriedade é declarada como uma variável do tipo booleana e define se o clipe de vídeo exibido pelo componente será executado em uma thread independente (quando seu valor for igual a False).

Componentes aos quais se aplica:

Na fase de projeto:

```
TAnimate
```

Durante a execução do aplicativo:

```
TAnimate
```

Title

Descrição

Para componentes do tipo TApplication, a propriedade Title é declarada como uma variável do tipo string e define o nome que será exibido com o ícone da aplicação quando ela for minimizada. Essa

propriedade só está disponível durante a execução do aplicativo. Você também pode alterar a propriedade Title de uma aplicação na página Application do quadro de diálogo Project Options do menu Options.

Para componentes do tipo TPrinter, a propriedade Title é declarada como uma variável do tipo string e define o nome que será exibido no gerenciador de impressão do Windows durante a impressão do documento. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

O trecho de código a seguir define o valor da propriedade Title de uma aplicação:

```
Application.Title := 'Nome';
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TApplication e TPrinter
```

TitleBeforeHeader

Descrição

Essa propriedade é declarada como uma variável booleana que define se o título do primeiro componente do tipo TQRBando será exibido ou não antes do cabeçalho da página.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
QuickReport1.TitleBeforeHeader:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TQuickReport
```

Durante a execução do aplicativo:

```
TQuickReport
```

TitleFont

Descrição

Essa propriedade é declarada como uma variável do tipo TFont e determina o tipo de fonte usada para exibir os títulos das colunas da grade.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou por meio da seleção de uma fonte em uma caixa de diálogo, com o seguinte trecho de código:

```
if FontDialog1.Execute then  DBGrid1.TitleFont := FontDialog1.Font;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBGrid
```

Durante a execução do aplicativo:

```
TDBGrid
```

Top

Descrição

A propriedade Top é uma variável inteira que define, em pixels, a coordenada da extremidade superior de um componente em relação à extremidade superior do formulário que o contém. No caso de um formulário, essa propriedade é medida em relação à tela.

Exemplo

Para alterar a propriedade Top de um botão chamado Button1 durante a execução de um aplicativo, basta incluir a seguinte linha de código no evento correspondente:

```
Button1.Top := valor;
```

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles.

Durante a execução do aplicativo:

Todos os controles e componentes TFindDialog e TReplceDialog.

ToPage

Descrição

A propriedade ToPage é uma variável inteira que define o número da última página a ser impressa.

Exemplo

Você pode alterar o valor dessa propriedade diretamente no Object Inspector ou mediante a inclusão de uma linha de código como:

```
PrintDialog1.ToPage:= 100;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TPrintDialog
```

Durante a execução do aplicativo:

```
TPrintDialog e TQRPrinter
```

TopAxis

Descrição

A propriedade TopAxis é declarada como um objeto da classe TChartAxis e representa o eixo horizontal superior do gráfico exibido no componente.

Componentes aos quais se aplica:

Na fase de projeto:

```
TChart, TDBChart
```

Durante a execução do aplicativo:

```
TChart, TDBChart
```

TopIndex

Descrição

A propriedade TopIndex é uma variável inteira que define o índice do item que aparece no topo de uma caixa de listagem. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode usar um componente chamado Edit1 do tipo TEdit para exibir o item exibido no topo de uma caixa de listagem chamada ListBox1 do tipo TListBox com a seguinte linha de código:

```
Edit1.Text := IntToStr(ListBox1.TopIndex);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDirectoryListBox, TFileListBox e TListBox
```

TopItem

Descrição

Para componentes do tipo TOutlineNode, essa propriedade é declarada como uma variável do tipo inteiro longo (Longint) que define o índice do seu ancestral cuja propriedade Level é igual a 1. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Para componentes do tipo TListView, essa propriedade é declarada como uma variável do tipo TListItem que define o item visível no topo do componente. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Para componentes do tipo TTreeView, essa propriedade é declarada como uma variável do tipo TTreeNode que define o item visível no topo do componente. Essa propriedade só está disponível durante a execução do aplicativo.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TListView, TTreeView e TOutlineNode
```

TopRow

Descrição

A propriedade TopRow é uma variável do tipo inteiro longo (Longint) que define a linha de células da grade que deve ser exibida no topo, imediatamente abaixo das linhas fixas.

Exemplo

Você pode alterar a propriedade TopRow diretamente no Object Inspector ou com a seguinte linha de código, na qual StringGrid1 é um componente do tipo TStringGrid1:

```
StringGrid1.TopRow := 1;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDrawGrid e TStringGrid
```

Durante a execução do aplicativo:

```
TDrawGrid e TStringGrid
```

TraceFlags

Descrição

Esta propriedade é declarada como uma variável do tipo TTraceFlag que define as operações a serem monitoradas pelo SQL Monitor.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDatabase, TIBDatabase
```

TrackLength

Descrição

A propriedade TrackLength é declarada com um array de inteiros longos (Longint), no qual cada item do array define o comprimento de uma trilha a ser executada por um dispositivo multimídia. Essa propriedade só está disponível durante a execução de um aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe o comprimento da segunda trilha a ser executada por um componente chamado MediaPlayer1 em um dispositivo multimídia.

```
Label1.Caption := IntToStr(MediaPlayer1.TrackLength[1]);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

TrackPosition

Descrição

A propriedade TrackPosition é declarada com uma array de inteiros longos (Longint) e retorna a posição inicial da trilha especificada pelo índice TrackNum.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe a posição inicial da segunda trilha a ser executada por um componente chamado MediaPlayer1 em um dispositivo multimídia.

```
Label1.Caption := IntToStr(MediaPlayer1.TrackPosition[1]);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

Tracks

Descrição

A propriedade Tracks é declarada com uma variável inteira longa (Longint) e retorna o número de trilhas que podem ser executadas no dispositivo multimídia corrente. Essa propriedade só está disponível durante a execução de um aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe o número de trilhas existentes em um componente chamado MediaPlayer1 em um dispositivo multimídia.

```
Label1.Caption := IntToStr(MediaPlayer1.Tracks);
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

TransactionCount

Descrição

Esta propriedade é declarada como uma variável inteira, e define o número de transações associadas ao componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBDatabase
```

Transactions

Descrição

Esta propriedade é declarada como uma array de objetos da classe TIBTransaction e permite acessar, através de um índice, um dos objetos TIBTransaction associados ao componente.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TIBDatabase
```

Transisolation

Descrição

A propriedade Transisolation é declarada como uma variável do tipo TTransisolation e define o nível do isolamento das transações com um servidor SQL.

Exemplo

Você pode definir o valor dessa propriedade diretamente no Object Inspector ou mediante uma linha de código como:

```
Database1.Transisolation := ReadCommited;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDatabase
```

Durante a execução do aplicativo:

```
TDatabase
```

Transliterate

Descrição

A propriedade Transliterate é uma variável booleana que define se é feita alguma transformação ao se passar um dado de um componente para outro.

Exemplo

Você pode definir o valor dessa propriedade mediante uma linha de código como:

```
BatchMove1.Transliterate := False;
```

onde BatchMove é um componente do tipo TBatchMove.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TBatchMove, TMemoField e TStringField
```

Transparent

Descrição

Essa propriedade é declarada como uma variável booleana que define se o componente será exibido com a cor definida na sua propriedade Color ou com a cor do componente sobre o qual está posicionado.

Exemplo

Você pode definir a propriedade Transparent de um componente diretamente no Object Inspector ou mediante uma linha de código. Para tornar um componente chamado Label1 do tipo TLabel transparente durante a execução de um aplicativo, basta incluir a seguinte linha de código:

```
Label1.Transparent := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TAnimate, TQRLabel, TLabel e TDBText
```

Durante a execução do aplicativo:

```
TAnimate, TQRLabel, TLabel e TDBText
```

Unidirecional

Descrição

A propriedade Unidirecional é uma variável booleana que define se a movimentação pelos registros de um conjunto de dados só pode ser feita em uma direção.

Exemplo

Você pode definir a propriedade Unidirecional de um componente diretamente no Object Inspector ou mediante uma linha de código como:

```
Query1.Unidirecional := False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TQuery, TDecisionQuery
```

Durante a execução do aplicativo:

```
TQuery, TDecisionQuery
```

UnSelectedColor

Descrição

A propriedade UnSelectedColor é declarada como uma variável do tipo TColor e define a cor das guias que não estão selecionadas em um componente do tipo TTabSet.

Exemplo

Você pode alterar a propriedade UnSelectedColor de um componente diretamente no Object Inspector ou por meio de uma linha de código, como:

```
TabSet1. UnSelectedColor := clBlue;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TTabSet
```

Durante a execução do aplicativo:

```
TTabSet
```

UpdatedMode

Descrição

A propriedade UpdatedMode é uma variável que define como o Delphi pesquisará os registros alterados em um banco de dados SQL.

Tabela de Valores:

Valor	Significado
WhereAll	A pesquisa é feita por todas as colunas.
WhereKeyOnly	A pesquisa é feita pelas colunas-chave.
WhereChanged	A pesquisa é feita pelas colunas-chave e as que já sofreram alteração.

Exemplo

Você pode definir a propriedade UpdatedMode de um componente diretamente no Object Inspector ou mediante uma linha de código como:

```
Query1.UpdatedMode := WhereAll;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TQuery, TDecisionQuery
```

Durante a execução do aplicativo:

```
TQuery, TDecisionQuery
```

UseCompression

Descrição

A propriedade UseCompression é uma variável do tipo booleano que define se deve-se ou não aplicar compressão ao relatório gerado.

Componentes aos quais se aplica:

Na fase de projeto:

```
TRenderPDF.
```

Durante a execução do aplicativo:

```
TRenderPDF.
```

Value

Descrição

Para componentes do tipo TDBRadioGroup, essa propriedade é declarada como uma variável do tipo string e armazena o conteúdo do campo correspondente no registro do banco de dados associado. A propriedade Items desse componente armazena uma lista de strings, na qual a primeira string corresponde ao primeiro botão de rádio, a segunda string ao segundo botão de rádio, e assim por diante. Quando o usuário seleciona um dos botões de rádio do grupo, a string correspondente é armazenada no campo correspondente do registro corrente do banco de dados associado.

Para componentes dos tipos TDBLookupCombo e TDBLookupList, essa propriedade é declarada como uma variável do tipo string e armazena o conteúdo do campo correspondente no registro do banco de dados associado. Para esses componentes, essa propriedade só está disponível durante a execução do aplicativo.

Nos demais casos, em componentes derivados de TField, armazena o valor correspondente ao campo que ele representa em registro de um banco de dados. Nesses casos, a variável será declarada como de um tipo compatível com o campo a ser representado, ou seja, será declarada como uma variável do tipo:

- String, para componentes do tipo TStringField.
- Longint, para componentes dos tipos TAutoIncField, TIntegerField, TSmallintField e TWordField.
- Double, para componentes dos tipos TBCDField, TCurrencyField e TFloatField.
- Boolean, para componentes do tipo TBooleanField.

TDateTime, para componentes dos tipos TDateField, TDateTimeField e TTimeField.

Para esses componentes, essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Durante a execução do aplicativo, você pode atribuir uma data à propriedade Value de um componente do tipo TDateField com a seguinte linha de código:

```
DateField1.Value:= StrToDateTime('01/15/96 10:30:00');
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TAutoIncField, TBCDField, TBooleanField, TCurrencyField, TDateField, TDateTimeField,
TDBLookupCombo, TDBLookupList, TFloatField, TIntegerField, TDBRadioGroup. TSmallintField,
TStringField, TTimeField e TWordField
```

ValueChecked

Descrição

A propriedade ValueChecked é uma variável do tipo string cujo valor será atribuído a um campo (definido pela propriedade DataField de TDBCheckBox) do registro corrente do banco de dados (definido pela propriedade DataSource de TDBCheckBox) quando o usuário selecionar a opção correspondente ao controle.

De modo inverso, se no campo do banco de dados for atribuído um valor diferente do armazenado em ValueChecked, a opção será desmarcada (o controle não será mais selecionado). Nesse caso, se a string armazenada no campo também não for igual à definida na propriedade ValueUnchecked, o controle ficará com aspecto acinzentado.

Você pode definir mais de uma string para a propriedade, desde que separadas por um ponto-e-vírgula. Nesse caso, basta que uma delas corresponda ao valor armazenado no campo para que o controle seja selecionado.

Exemplo

Você pode alterar a propriedade ValueChecked diretamente no Object Inspector ou com a seguinte linha de código:

```
DBCheckBox1.ValueChecked := 'Valor';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBCheckBox
```

Durante a execução do aplicativo:

```
TDBCheckBox
```

Values

Descrição

A propriedade Values é declarada como uma variável do tipo TStrings que armazena uma lista de strings em que cada item corresponderá a um possível valor de um campo em um registro do banco de dados associado. Cada item dessa lista corresponde a um item da lista de strings armazenada na propriedade Items do componente. O primeiro item corresponderá ao primeiro botão de rádio, o segundo item ao segundo botão de rádio, e assim por diante. O conteúdo da propriedade Values é que será comparado com o valor corrente do campo, isto é, se uma das strings da propriedade Values coincidir com o armazenado no campo do banco de dados, o botão de rádio correspondente será selecionado. De maneira inversa, se um botão de rádio for selecionado, a string correspondente na propriedade Values será armazenada no campo de dados (desde que a propriedade Read Only seja False). Caso seja omitida a lista de strings na propriedade Values, será considerada a lista armazenada na propriedade Items. Você pode operar sobre a propriedade Values da mesma forma que opera sobre qualquer item de uma lista de strings, para adicionar, remover ou inserir uma string.

Para objetos dos tipos TStrings e TStringList, dá acesso direto a uma lista de strings.

Exemplo

Você pode alterar a propriedade Values diretamente no Object Inspector ou mediante um trecho de código, como exemplificado a seguir. Nesse caso, quando o formulário é criado, atribui-se à propriedade Items de um componente DBRadioGroup1, do tipo TDBRadioGroup, a lista de strings composta por

'Vermelho', 'Verde' e 'Amarelo', e à sua propriedade Values, a lista de strings composta por 'Red', 'Green' e 'Yellow'.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 with DBRadioGroup1 do
 begin
   Items.Add('Vermelho');
   Items.Add('Verde');
   Items.Add('Amarelo');
   Values.Add('Red');
   Values.Add('Green');
   Values.Add('Yellow');
 end;
end;
```

Quando o aplicativo é executado, surgem três botões de rádio no componente DBRadioGroup, com rótulos iguais a Vermelho, Verde e Amarelo. Se o valor armazenado no campo do registro correspondente ao banco de dados associado for igual a Red, Green ou Blue, o botão correspondente será selecionado. Por outro lado, se o usuário selecionar um dos botões de rádio, a string correspondente na propriedade Values será armazenada no campo do banco de dados.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBRadioGroup
```

Durante a execução do aplicativo:

```
TStrings, TStringList e TDBRadioGroup
```

ValueUnChecked

Descrição

A propriedade ValueUnChecked é uma variável do tipo string cujo valor será atribuído a um campo (definido pela propriedade DataField de TDBCheckBox) do registro corrente do banco de dados (definido pela propriedade DataSource de TDBCheckBox) quando o usuário desfizer a seleção da opção correspondente ao controle.

De modo inverso, se no campo do banco de dados for atribuído um valor diferente do armazenado em ValueUnChecked, a opção será desmarcada (o controle não será mais selecionado). Nesse caso, se a string armazenada no campo também não for igual à definida na propriedade ValueChecked, o controle ficará com aspecto acinzentado.

Você pode definir mais de uma string para a propriedade, desde que separadas por um ponto-e-vírgula. Nesse caso, basta que uma delas corresponda ao valor armazenado no campo para que o controle não seja selecionado.

Exemplo

Você pode alterar a propriedade ValueUnChecked diretamente no Object Inspector ou com a seguinte linha de código:

```
DBCheckBox1.ValueUnChecked := 'Valor';
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBCheckBox
```

Durante a execução do aplicativo:

```
TDBCheckBox
```

VERTSCROLLBAR

Descrição

A propriedade VertScrollBar é declarada como uma variável do tipo TControlScrollBar cujas subpropriedades definem o comportamento de uma barra de rolagem vertical em um componente do tipo TForm, TFrame ou TScrollBox.

Exemplo

Você pode alterar as subpropriedades da propriedade VertScrollBar de um componente diretamente no Object Inspector ou durante a execução do aplicativo incluindo uma linha de código como:

```
Form1.VertScrollBar.Visible := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm, TFrame e TScrollBox
```

Durante a execução do aplicativo:

```
TForm e TScrollBox
```

VIEWORIGIN

Descrição

Essa propriedade é declarada como uma variável do tipo TPoint e retorna as coordenadas da origem do controle. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TListView
```

VIEWSTYLE

Descrição

A propriedade ViewStyle é declarada como uma variável do tipo TViewStyle que define como os itens serão exibidos no controle.

Tabela de Valores:

Valor	Significado
vsIcon	Cada item aparece como um ícone em verdadeira grandeza sobre um rótulo. Os itens podem ser arrastados pelo usuário.
vsSmallIcon	Cada item aparece como um pequeno ícone com um rótulo à sua direita. Os itens podem ser arrastados pelo usuário.
vsList	Cada item aparece como um pequeno ícone com um rótulo à sua direita. Os itens são dispostos em colunas e não podem ser arrastados pelo usuário.
vsReport	Cada item é exibido em sua própria linha, com informações dispostas em colunas.

Componentes aos quais se aplica:

Na fase de projeto:

```
TListView
```

Durante a execução do aplicativo:

```
TListView
```

VISIBLE

Descrição

A propriedade Visible é uma variável booleana que define se o componente aparece ou não na tela.

Para objetos da classe TAction, define o valor da propriedade Visible dos controles e itens de menu associados ao objeto.

Exemplo

Você pode definir a propriedade Visible de um componente diretamente no Object Inspector ou mediante uma linha de código. Para tornar um botão Button1 invisível durante a execução de um aplicativo, basta incluir a seguinte linha de código:

```
Button1.Visible:= False;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
Todos os controles; componentes, TAction TBCDField, TBlobField, TBooleanField, TBytesField,
TControlScrollBar, TCurrencyField, TDateField, TDateTimeField, TFloatField, TForm,
TGraphicField, TIntegerField, TMenuItem, TMemoField, TSmallintField, TStringField, TTimeField,
TVarBytesField, TWordField
```

Durante a execução do aplicativo:

```
Todos os controles; componentes, TAction TBCDField, TBlobField, TBooleanField, TBytesField,
TControlScrollBar, TCurrencyField, TDateField, TDateTimeField, TFloatField, TForm,
TGraphicField, TIntegerField, TMenuItem, TMemoField, TSmallintField, TStringField, TTimeField,
TVarBytesField, TWordField
```

VISIBLEBUTTONS

Descrição

A propriedade VisibleButtons é declarada como uma variável do tipo TButtonSet que define os botões a serem exibidos por um controle dos tipos TDBNavigator ou TMediaPlayer.

Tabela de Valores para controles do tipo TDBNavigator.

Botão	Valor	Significado
First	nbFirst	Exibe o primeiro registro do banco de dados.
Prior	nbPrior	Exibe o registro anterior do banco de dados.
Next	nbNext	Exibe o próximo registro do banco de dados.
Last	nbLast	Exibe o último registro do banco de dados.
Insert	nbInsert	Insere um registro em branco no banco de dados.
Delete	nbDelete	Deleta o registro corrente do banco de dados.

continua

continuação

Botão	Valor	Significado
Edit	nbEdit	Permite a edição do registro corrente do banco de dados.
Post	nbPost	Grava o registro corrente no banco de dados.
Cancel	nbCancel	Cancela a edição do registro corrente do banco de dados.
Refresh	nbRefresh	Atualiza a exibição dos registros do banco de dados.

Tabela de Valores para controles do tipo TMediaPlayer.

Botão	Valor	Significado
Play	btPlay	Inicia reprodução.
Record	btRecord	Inicia gravação.
Stop	btStop	Interrompe uma reprodução ou gravação.
Next	btNext	Passa para a trilha seguinte.
Prev	btPrev	Passa para a trilha seguinte.
Step	btStep	Desloca um certo número de quadros para frente.
Back	btBack	Desloca um certo número de quadros para frente.
Pause	btPause	Pausa durante a reprodução ou gravação.
Eject	btEject	Ejeção.

Exemplo

Os valores das subpropriedades da propriedade VisibleButtons do controle podem ser alterados diretamente no Object Inspector.

Caso queira alterá-las durante a execução do aplicativo, as subpropriedades devem ser listadas entre colchetes, separadas por vírgulas e, então, o conjunto deve ser atribuído à propriedade VisibleButtons, como na linha de código abaixo:

```
procedure TForm1.FormDblClick(Sender: TObject);
begin
   DBNavigator1.VisibleButtons := [nbFirst,nbPrior,nbNext,nbLast];
end;
```

Nesse caso, ao se dar um duplo clique com o mouse sobre um formulário chamado Form1, será redefinida a propriedade VisibleButtons do controle DBNavigator1, do tipo TDBNavigator.

As subpropriedades listadas entre colchetes terão o valor True e as demais, o valor False, independentemente do valor que lhes tenha sido atribuído no Object Inspector.

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBNavigator e TMediaPlayer
```

Durante a execução do aplicativo:

```
TDBNavigator e TMediaPlayer
```

VisibleColCount

Descrição

A propriedade VisibleColCount é uma variável inteira que define o número de colunas (exceto as fixas) que são integralmente exibidas pela grade. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe o número de colunas integralmente exibidas por um componente StringGrid1 do tipo TStringGrid quando o usuário seleciona uma célula com o mouse.

```
procedure TForm1.StringGrid1Click(Sender: TObject);
begin
  Label1.Caption := IntToStr(StringGrid1.VisibleColCount);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TDrawGrid e TStringGrid
```

VisibleRowCount

Descrição

Para componentes dos tipos TDrawGrid e TStringGrid, a propriedade VisibleRowCount é uma variável inteira que define o número de linhas (exceto as fixas) integralmente exibidas pela grade. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Para componentes do tipo TListView, a propriedade VisibleRowCount é uma variável inteira que define o número de itens que podem ser exibidos na área visível do controle. Essa propriedade só está disponível durante a execução do aplicativo, e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe o número de linhas integralmente exibidas por um componente StringGrid1 do tipo TStringGrid quando o usuário seleciona uma célula com o mouse.

```
procedure TForm1.StringGrid1Click(Sender: TObject);
begin
     Label1.Caption:= IntToStr(StringGrid1.VisibleRowCount);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TListView, TDrawGrid e TStringGrid
```

VisibleTabs

Descrição

A propriedade VisibleTabs é uma variável inteira que define o número de guias visíveis no controle. Essa propriedade só está disponível durante a execução do aplicativo e não pode ter o seu valor diretamente alterado pelo usuário.

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel informe o número de guias visíveis em um componente TabSet1 do tipo TTabSet quando o usuário seleciona uma guia do controle com o mouse.

```
procedure TForm1.TabSet1Click(Sender: TObject);
begin
   Label1.Caption := IntToStr(TabSet1.VisibleTabs);
end;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TTabSet
```

Wait

Descrição

A propriedade Wait é uma variável booleana que define se um método de controle do tipo TMediaPlayer só deve retornar o controle para a aplicação após encerrar a sua execução. Essa propriedade só está disponível durante a execução do aplicativo.

Exemplo

Você pode alterar o valor da propriedade Wait mediante uma linha de código como:

```
MediaPlayer1,Wait := False;
```

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TMediaPlayer
```

WantReturns

Descrição

A propriedade WantReturns é uma variável booleana que define se a tecla Enter está habilitada em um componente.

Exemplo

Você pode definir a propriedade WantReturns de um componente diretamente no Object Inspector ou mediante uma linha de código. Para habilitar a tecla Enter em um controle durante a execução de um aplicativo, basta incluir a seguinte linha de código:

```
Memo1.WantsReturns:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMemo, TDBMemo e TRichEdit
```

Durante a execução do aplicativo:

```
TMemo, TDBMemo e TRichEdit
```

WantTabs

Descrição

A propriedade WantTabs é uma variável booleana que define se a tabulação está habilitada em um componente.

Exemplo

Você pode definir a propriedade WantTabs de um componente diretamente no Object Inspector ou mediante uma linha de código. Para habilitar a tabulação em um controle durante a execução de um aplicativo, basta incluir a seguinte linha de código:

```
Memo1.WantsTabs := True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TMemo e TDBMemo
```

Durante a execução do aplicativo:

```
TMemo e TDBMemo
```

Width

Descrição

A propriedade Width é uma variável inteira que define a dimensão horizontal, em pixels, de um controle ou componente gráfico. Para componentes do tipo TScreen, retorna a largura da tela, em pixels. Para objetos gráficos, define a sua espessura em Pixels.

Exemplo

Para alterar a dimensão horizontal de um formulário chamado Form1 durante a execução de um aplicativo, basta incluir a seguinte linha de código no evento correspondente:

```
Form1.Width := valor;
```

Componentes aos quais se aplica:

Na fase de projeto:

Todos os controles.

Durante a execução do aplicativo:

Todos os controles; TBitmap, TFont, TGraphic, TIcon, TMetafile, TPen e TPicture.

WindowMenu

Descrição

A propriedade WindowMenu é declarada como uma variável do tipo TMenuItem que define o menu que manipulará as janelas-filhas de uma aplicação MDI.

Exemplo

Você pode definir a propriedade WindowMenu de um formulário diretamente no Object Inspector ou mediante uma linha de código, como:

```
Form1.WindowMenu := Janelas;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TForm
```

WINDOWSTATE

Descrição

A propriedade WindowState é declarada como uma variável do tipo TWindowState que define o estado de exibição de um formulário.

Tabela de Valores:

Valor	Significado
wsNormal	O formulário não está minimizado ou maximizado.
wsMaximized	O formulário está maximizado.
wsMinimized	O formulário está minimizado.

Exemplo

Você pode definir a propriedade WindowState de um formulário diretamente no Object Inspector ou mediante uma linha de código, como:

```
Form1.WindowState := wsNormal;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TForm
```

Durante a execução do aplicativo:

```
TForm
```

WORDWRAP

Descrição

A propriedade WordWrap é uma variável booleana que define se o texto digitado deve passar para a linha seguinte quando atingir a margem direita do controle.

Exemplo

Você pode definir a propriedade WordWrap de um componente diretamente no Object Inspector ou mediante uma linha de código, como:

```
Memo1.WordWrap:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TDBMemo, TDBText, TLabel, TRichEdit e TMemo
```

Durante a execução do aplicativo:

```
TDBMemo, TDBText, TLabel, TRichEdit e TMemo
```

Wrap

Descrição

A propriedade Wrap é uma variável booleana que define se, caso o valor armazenado na propriedade Position ultrapasse o valor definido na propriedade Max, Position assume o valor definido na propriedade Min.

Exemplo

Você pode definir a propriedade Wrap de um componente diretamente no Object Inspector ou mediante uma linha de código, como:

```
UpDown1.Wrap:= True;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TUpDown
```

Durante a execução do aplicativo:

```
TUpDown
```

WSAInfo

Descrição

Essa propriedade é declarada como um objeto da classe TStringList e armazena informações sobre a versão do WinSock usada em uma conexão.

Componentes aos quais se aplica:

Durante a execução do aplicativo:

```
TNMEcho, TNMFinger, TNMFTP, TNMHTTP, TNMMSG, TNMMSGServ, TNMNNTP, TNMPOP3, TNMSMTP, TNMSTRM,
TNMSTRMServ e TPowerSock
```

Zoom

Descrição

Para componentes do tipo TOLEContainer, a propriedade Zoom é declarada como uma variável do tipo TZoomFactor que define o fator de amplificação ou redução a ser aplicado em um objeto OLE a ser exibido dentro do componente.

Para componentes dos tipos TQRPreview, a propriedade Zoom é declarada como uma variável inteira que especifica o nível de zoom (em percentagem) aplicado sobre a imagem.

Tabela de Valores:

Valor	Significado
z025	O objeto OLE é exibido com 25% do seu tamanho original.
z050	O objeto OLE é exibido com 50% do seu tamanho original.
z100	O objeto OLE é exibido com seu tamanho original.
z150	O objeto OLE é exibido com 150% do seu tamanho original.
z200	O objeto OLE é exibido com 200% do seu tamanho original.

Se o objeto OLE ficar maior do que o componente do tipo TOLEContainer que o contém, ocorrerá um clipping (recorte) da imagem, mas o objeto não será alterado – apenas a sua exibição será alterada.

Exemplo

Você pode definir a propriedade Zoom de um componente do tipo TOLEContainer diretamente no Object Inspector ou mediante uma linha de código, como:

```
OLEContainer1.Zoom:= z150;
```

Componentes aos quais se aplica:

Na fase de projeto:

```
TOLEContainer e TQRPreview
```

Durante a execução do aplicativo:

```
TOLEContainer e TQRPreview
```

Capítulo 42

Métodos

Abort

Descrição

O método Abort interrompe o trabalho de impressão corrente.

Declaração

```
procedure Abort;
```

Componentes aos quais se aplica:

```
TPrinter
```

Activate

Descrição

O método Activate garante que este componente será o primeiro a responder aos eventos gerados na aplicação.

Declaração

```
procedure Activate;
```

Componentes aos quais se aplica:

```
TApplicationEvents
```

Add

Descrição

O método Add adiciona um item a uma lista de objetos e, para objetos dos tipos TList, TStrings, TStringList e TOutline, retorna a posição do item na lista. O tipo de item depende dos objetos armazenados na lista.

Declaração

A declaração depende do tipo de item armazenado na lista.

Para objetos do tipo TList:

```
function Add(Item: Pointer): Integer;
```

Para objetos dos tipos TStrings e TStringList:

```
function Add(const S: string): Integer;
```

Para objetos do tipo TMenuItem:

```
procedure Add(Item: TMenuItem);
```

Para objetos do tipo TOutline:

```
function Add(Index: LongInt; const Text: string): LongInt;
```

Para objetos do tipo TFieldDefs:

```
procedure Add(const Name: string; DataType: TFieldType; Size: Word);
```

Para objetos do tipo TIndexDefs:

```
procedure Add(const Name, Fields: string; Options: TIndexOptions);
```

Exemplo

O trecho de código a seguir acrescenta um item a uma lista de strings que representa a propriedade Items de um componente chamado ListBox1 do tipo TListBox e retorna a posição do item em uma variável chamada Position:

```
Position:= ListBox1.Items.Add('Novo item');
```

Componentes aos quais se aplica:

```
TList, TStrings, TStringList, TMenuItem, TOutline, TFieldDefs e TIndexDefs
```

AddChild

Descrição

O método AddChild adiciona um item como um subitem do item definido no parâmetro Index em um componente do tipo TOutline.

Declaração

```
function AddChild(Index: LongInt; const Text:
string): LongInt;
```

Exemplo

O trecho de código a seguir acrescenta um item como subitem do item selecionado em um componente chamado Outline1 do tipo TOutline:

```
Outline1.AddChild(Outline1.SelectedItem, 'Novo item');
```

Componentes aos quais se aplica:

```
TOutline
```

AddChildObject

Descrição

O método AddChildObject adiciona um item contendo dados como um subitem do item definido no parâmetro Index em um componente do tipo TOutline.

Declaração

```
function AddChildObject(Index: LongInt; const Text: string; const Data: Pointer): LongInt;
```

Exemplo

Veja o exemplo do método AddChild.

Componentes aos quais se aplica:

```
TOutline
```

AddDatabase

Descrição

Esse método associa um banco de dados, representado por um objeto da classe TIBDatabase, à transação corrente.

Declaração

```
function AddDatabase(db: TIBDatabase): Integer;
```

Componentes aos quais se aplica:

```
TIBTransaction
```

AddDataset

Descrição

Esse método associa um banco de dados, representado por um objeto da classe TIBDatabase, à transação corrente.

Declaração

```
function AddDatabase(db: TIBDatabase): Integer;
```

Componentes aos quais se aplica:

```
TIBTransaction
```

AddIcon

Descrição

Esse método adiciona um ícone a um objeto do tipo TImageList.

Declaração

```
function AddIcon(Image: TIcon): Integer;
```

Componentes aos quais se aplica:

```
TImageList
```

AddIndex

Descrição

O método AddIndex adiciona um índice à tabela.

Declaração

```
procedure AddIndex(const Name, Fields: string; Options: TIndexOptions);
```

Componentes aos quais se aplica:

```
TClientDataSet, TSimpleDataset, TIBClientDataset, TIBTable, TTable
```

AddMasked

Descrição

Esse método adiciona um ícone a um objeto do tipo TImageList e uma cor usada como máscara transparente (passada pelo parâmetro MaskColor).

Declaração

```
function AddMasked(Image: TBitmap; MaskColor: TColor): Integer;
```

Componentes aos quais se aplica:

```
TImageList
```

AddObject

Descrição

Para objetos do tipo TOutline, esse método adiciona um item contendo dados ao componente.

Para componentes dos tipos TStrings e TStringList, adiciona a string definida no parâmetro S, e o objeto definido no parâmetro AObject.

Declaração

Para componentes do tipo TOutline: function AddObject(Index: LongInt; const Text: string; const Data: Pointer): LongInt;

Para componentes dos tipos TStrings e TStringList: function AddObject(const S: string; AObject: TObject): Integer;

Componentes aos quais se aplica:

```
TStrings, TStringList e TOutline
```

AddPassword

Descrição

O método AddPassword é usado para adicionar uma nova senha, definida pelo parâmetro Password, ao componente do tipo TSession para uso com tabelas do Paradox.

Declaração

```
procedure AddPassword(const Password: string);
```

Exemplo

O trecho de código a seguir adiciona uma senha a um componente do tipo TSession:

```
Session.AddPassword('Senha');
```

Componentes aos quais se aplica:

TSession

AddScriptFile

Descrição

Esse método adiciona um arquivo de script, cujo nome é passado como parâmetro, à Applet representada pelo componente.

Declaração

```
procedure AddScriptFile(const AFilename: String);
```

Componentes aos quais se aplica:

TIWApplet, TIWButton, TIWCheckbox, TIWCombobox, TIWControl, TIWDBCheckbox, TIWDBEdit, TIWDBFile, TIWDBGrid, TIWDBImage, TIWDBListbox, TIWDBLookupCombobox, TIWDBLookupListbox, TIWDBMemo, TIWDBNavigator, TIWDBText, TIWEdit, TIWForm, TIWGrid, TIWImage, TIWImageFile, TIWLabel, TIWLink, TIWList, TIWListbox, TIWMemo, TIWRectangle, TIWTimer, TIWTreeview e TIWURL

AddSeries

Descrição

Esse método adiciona uma série ao gráfico exibido no componente.

Declaração

```
procedure AddSeries(ASeries: TChartSeries);
```

Componentes aos quais se aplica:

TChart e TDBChart

AddStrings

Descrição

Esse método adiciona um grupo de strings, definido pelo parâmetro Strings, à lista mantida pelo objeto.

Declaração

```
procedure AddStrings(Strings: TStrings);
```

Componentes aos quais se aplica:

TStrings e TStringList

AddTransaction

Descrição

Esse método associa uma transação representada por um componente TIBTransaction e passada como parâmetro a um banco de dados representado pelo componente.

Declaração

```
function AddTransaction(TR: TIBTransaction): Integer;
```

Componentes aos quais se aplica:

TIBDatabase

AddXY

Descrição

O método AddXY adiciona um ponto à série representada pelo objeto.

Declaração

```
function AddXY(Const AXValue, AYValue: Double; Const AXLabel: String; AColor: TColor):
Longint;Exemplo
```

SComponentes aos quais se aplica:

```
TChartSeries
```

AddY

Descrição

O método AddY adiciona um ponto à série representada pelo objeto, quando os pontos não possuem uma abscissa (ex.: gráfico de torta).

Declaração

```
function AddY(Const AYValue: Double; Const AXLabel: String; AColor: TColor): LongInt;
```

Componentes aos quais se aplica:

```
TChartSeries
```

Append

Descrição

Esse método adiciona um registro após o último registro existente em um banco de dados, depois de colocá-lo no modo de inserção e chamar o seu método Post.

Declaração

```
procedure Append;
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc,TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TSimpleDataset, TSQLDataset, TSQLTable, TSQLQuery
e TSQLStoredProc
```

AppendRecord

Descrição

Esse método adiciona um registro após o último registro existente em um banco de dados, usando os valores passados pelo parâmetro Values.

Declaração

```
procedure AppendRecord(const Values: array of const);
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc,TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TSimpleDataset, TSQLDataset, TSQLTable, TSQLQuery
e TSQLStoredProc
```

ApplyRange

Descrição

O método ApplyRange é usado para aplicar os valores definidos pelos métodos SetRangeStart, SetRangeEnd, EditRangeStart e EditRangeEnd à tabela.

Declaração

```
procedure ApplyRange;
```

Componentes aos quais se aplica:

```
TClientDataSet, TSimpleDataset e TTable
```

ApplyUpdates

Descrição

O método ApplyUpdates é usado para atualizar os valores de uma tabela com os armazenados no seu cache.

Declaração

```
procedure ApplyUpdates;
```

Componentes aos quais se aplica:

```
TADOQuery, TIBDataset, TIBQuery, TIBTransaction, TClientDataset, TIBClientDataset,
TSimpleDataset e TQuery
```

Arc

Descrição

O método Arc desenha um arco no perímetro da elipse limitada pelo retângulo definido pelas coordenadas (X1,Y1) e (X2,Y2). O arco começa no ponto de interseção da linha que vai do centro da elipse ao ponto (X3,Y3) com o retângulo e vai até o ponto em que o retângulo intercepta a linha que vai do centro da elipse ao ponto (X4,Y4). O desenho é feito no sentido anti-horário.

Declaração

```
procedure Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
```

Componentes aos quais se aplica:

```
TCanvas
```

ArrangeIcons

Descrição

Em aplicações MDI, esse método organiza as janelas-filhas que estão minimizadas, de forma que fiquem igualmente espaçadas e não se sobreponham.

Declaração

```
procedure ArrangeIcons;
```

Exemplo

O trecho de código a seguir faz com que o método ArrangeIcons seja acionado quando o usuário seleciona um item de menu Window OrganizarIcones:

```
procedure TForm1.WindowOrganizarIconesClick(Sender: TObject);
begin
     Form1.ArrangeIcons;
end;
```

Componentes aos quais se aplica:

```
TForm
```

Assign

Descrição

Para componentes dos tipos TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField, TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField, TStringField, TTimeField, TVarBytesField e TWordField, o método Assign copia o valor de um campo (definido pelo parâmetro Source) em outro (desde que sejam compatíveis).

Para objetos do tipo TClipBoard, copia o objeto definido pelo parâmetro Source no Clipboard.

Para objetos do tipo TIndexDefs, copia os objetos definidos na propriedade Index do parâmetro IndexDefs.

Para objetos do tipo TParams, transfere as informações do parâmetro Params.

Para os demais tipos de objetos, atribui um objeto a outro.

Declaração

Para objetos do tipo TIndexDefs:

```
procedure Assign(IndexDefs: TIndexDefs);
```

Para objetos do tipo TParams:

```
procedure Assign(Param: TParam);
```

Para os demais objetos:

```
procedure Assign(Source: TPersistent);
```

Exemplo

O trecho de código a seguir copia o texto armazenado em um componente Memo1 do tipo TMemo em um campo MemoField1 do tipo TMemoField:

```
MemoField1.Assign(Memo1.Lines);
```

Componentes aos quais se aplica:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TClipBoards, TCurrencyField,
TDateField, TDateTimeField, TFloatField, TGraphicField, TIndexDefs, TIntegerField, TMemoField,
TParams, TSmallintField, TStringField, TTimeField, TVarBytesField, TBitmap, TBrush,
TControlScrollBar, TFieldDef, TFont, TIcon, TIndexDef, TMetafile, TPen, TPicture, TStringList,
TStrings e TWordField
```

AssignValue

Descrição

O método AssignValue atribui um valor (especificado no parâmetro Value) a um campo.

Declaração

```
procedure AssignValue(const Value: TVarRec);
```

Exemplo

O trecho de código atribui uma string a um campo StringField1, do tipo TStringField:

```
StringField1.AssignValue('nova string atribuída ao valor');
```

Componentes aos quais se aplica:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

AtLeast

Descrição

Este método verifica se o número de elementos referenciados pela estrutura representada pela classe (que pode ser uma Fila ou uma Pilha) é no mínimo igual ao valor passado como parâmetro.

Declaração

```
function AtLeast(ACount: Integer): Boolean;
```

Componentes aos quais se aplica:

```
TQueue e TStack.
```

Back

Descrição

O método Back faz com que o dispositivo multimídia retorne um certo número de quadros, especificado na propriedade Frames.

Declaração

```
procedure Back;
```

Exemplo

O trecho de código a seguir faz com que um dispositivo multimídia acione seu método Back com um botão chamado Back do tipo TButton:

```
procedure TForm1.BackClick(Sender: TObject);
begin
      MediaPlayer1.Back;
end;
```

Componentes aos quais se aplica:

```
TMediaPlayer
```

BatchMove

Descrição

O método BatchMove é usado para copiar, atualizar, adicionar ou deletar registros de uma tabela representada pelo parâmetro ASource e de acordo com o parâmetro AMode, que pode ter um dos seguintes valores: batAppend, batUpdate, batAppendUpdate, batDelete ou batCopy.

Declaração

```
function BatchMove(ASource: TDataSet; AMode: TBatchMode): LongInt;
```

Componentes aos quais se aplica:

```
TClientDataSet, TTable
```

BeginDoc

Descrição

Para componentes do tipo TQRPrinter, esse método inicia o trabalho de impressão de um relatório e atribui o valor mbBusy à propriedade Status do componente.

Para componentes do tipo TPrinter, indica o início de um trabalho de impressão.

Declaração

```
procedure BeginDoc;
```

Componentes aos quais se aplica:

```
TPrinter e TQRPrinter
```

BeginDrag

Descrição

O método BeginDrag inicia o processo de arrastar o controle. Se o valor da variável Immediate é True, o processo de arrastar ocorre imediatamente. Se Immediate é False, o processo não se inicia até que o usuário mova o ponteiro do mouse a uma distância de aproximadamente 5 pixels. Isso permite que um controle aceite um clique de mouse antes de iniciar um processo de arrastar.

Observação: Você não precisa usar o método BeginDrag quando a propriedade DragMode do controle tem o valor dmAutomatic.

Declaração

```
procedure BeginDrag(Immediate: Boolean);
```

Exemplo

O trecho de código a seguir faz com que um botão chamado Button1 execute o método BeginDrag se a sua propriedade DragMode for dmManual L:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 if Button1.DragMode = dmManual then
   Button1.BeginDrag(True);
end;
```

Componentes aos quais se aplica:

Todos os controles.

BeginTrans

Declaração

Esse método inicializa uma nova transação com o banco de dados.

Descrição

function BeginTrans: Integer;

Componentes aos quais se aplica:

```
TADOConnection
```

BeginUpdate

Declaração

Esse método evita que a exibição de um objeto dos tipos TStrings, TStringList ou TOutline seja atualizada até que se chame o método EndUpdate.

Descrição

procedure BeginUpdate;

Exemplo

O trecho de código a seguir faz com que um componente chamado Outline1 execute o seu método BeginUpdate:

```
Outline1.BeginUpdate;
```

Componentes aos quais se aplica:

```
TStringList, TStrings e TOutline
```

BookMarkValid

Descrição

O método BookMarkValid é usado para verificar se existe um valor atribuído a um objeto da classe TBookmark passado como parâmetro.

Declaração

```
function BookmarkValid(Bookmark: TBookmark): Boolean; override;
```

Componentes aos quais se aplica:

```
TClientDataSet, TIBSQLClientDataset, TSimpleDataset e TTable
```

BringToFront

Descrição

O método BringToFront coloca o controle na frente de todos os outros controles que existem no mesmo formulário. Uma forma de garantir que um controle está visível durante a execução de um aplicativo é definir a sua propriedade Visible como True e utilizar o método BringToFront.

Declaração

```
procedure BringToFront;
```

Exemplo

O trecho de código a seguir faz com que um botão de rádio chamado RadioButton1 seja colocado na frente de todos os outros componentes que existem no mesmo formulário e que possam estar impedindo a sua visibilidade:

```
RadioButton1.BringToFront;
```

Componentes aos quais se aplica:

Todos os controles e componentes do tipo TForm.

Broadcast

Descrição

Esse método envia a mensagem definida no parâmetro Message para cada um dos controles-filhos do controle corrente.

Declaração

```
procedure Broadcast(var Message);
```

Componentes aos quais se aplica:

```
Todos os controles.
```

BrushCopy

Descrição

Esse método copia uma porção de um bitmap para o canvas. O parâmetro Dest representa a área retangular do canvas na qual a área retangular do bitmap será copiada. O parâmetro Bitmap representa o bitmap a ser copiado. O parâmetro Source representa a área retangular do bitmap. O parâmetro Color define a cor do bitmap a ser substituída pela cor do canvas.

Declaração

```
procedure BrushCopy(const Dest: TRect; Bitmap: TBitmap; const Source: TRect; Color: TColor);
```

Componentes aos quais se aplica:

```
TCanvas
```

Call

Descrição

Esse método retorna a mensagem de erro cujo código é passado como parâmetro.

Declaração

```
function Call(ErrCode: ISC_STATUS; RaiseError: Boolean): ISC_STATUS;
```

Componentes aos quais se aplica:

```
TapplicationEvents, TIBTransaction, TIBDatabase, TIBSQL
```

CanAutoSize

Descrição

Esse método atribui novos valores às dimensões dos quadros que formam o clipe de vídeo AVI exibido pelo componente.

Declaração

```
function CanAutoSize(var NewWidth, NewHeight: Integer): Boolean; override;
```

Componentes aos quais se aplica:

```
TAnimate
```

Cancel

Descrição

Para componentes derivados da classe TDataset, esse método cancela as alterações feitas em um banco de dados.

Para componentes do tipo TQRPrinter, esse método cancela o trabalho de impressão de um relatório e atribui o valor mbReady à propriedade Status do componente.

Declaração

```
procedure Cancel;
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc,TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TClientDataset, TIBSQLClientDataset e
TSimpleDataset
```

CancelDispatch

Descrição

Este método evita que outros objetos desta classe respondam aos eventos gerados pela aplicação.

Declaração

```
procedure CancelDispatch;
```

Componentes aos quais se aplica:

```
TApplicationEvents
```

CancelEvents

Descrição

Este método cancela a execução de eventos pendentes do servidor.

Declaração

```
procedure CancelEvents;
```

Componentes aos quais se aplica:

```
TIBEvents
```

CancelRange

Descrição

Esse método remove qualquer limitação de faixas anteriormente estabelecidas pelos métodos ApplyRange ou SetRange.

Declaração

```
procedure CancelRange;
```

Componentes aos quais se aplica:
```
TClientDataSet, TIBSQLClientDataset, TSimpleDataset e TTable
```

CancelUpdates

Descrição

O método CancelUpdates é usado para cancelar as modificações efetuadas nos valores de uma tabela, armazenados no seu cache.

Declaração
```
procedure CancelUpdates;
```

Componentes aos quais se aplica:
```
TClientDataSet, TIBSQLClientDataset, TSimpleDataset, TIBTable, TADOTable e TTable
```

CanFocus

Descrição

Esse método determina se o controle pode receber o foco da aplicação.

Declaração
```
function CanFocus: Boolean;
```

Exemplo

O trecho de código a seguir faz com que um botão chamado Button1 fique invisível se um botão chamado Button2 puder receber o foco:
```
procedure TForm1.Button2Click(Sender: TObject);
begin
     Button1.Visible := not CanFocus;
end;
```

Componentes aos quais se aplica:
```
Todos os controles.
```

Cascade

Descrição

Em aplicações MDI, esse método faz com que as janelas-filhas se sobreponham "em cascata", de forma que a barra de títulos das janelas permaneça visível e possa ser selecionada.

Declaração
```
procedure Cascade;
```

Exemplo

O trecho de código a seguir faz com que o método Cascade seja acionado quando o usuário seleciona um item de menu Windowl Cascata:
```
procedure TForm1.WindowCascataClick(Sender: TObject);
begin
      Form1.Cascade;
end;
```

Componentes aos quais se aplica:
```
TForm
```

CellRect

Descrição

Esse método retorna uma variável do tipo TRect para a célula definida pela linha e coluna especificadas pelos parâmetros ARow e ACol, respectivamente.

Declaração
```
function CellRect(ACol, ARow: LongInt): TRect;
```

Componentes aos quais se aplica:
```
TDrawGrid e TStringGrid
```

CHANGE

Descrição

Esse método executa o procedimento associado ao evento OnChange do componente.

Declaração
```
procedure Change; virtual;
```

Componentes aos quais se aplica:
```
TActionList
```

CHANGELEVELBY

Descrição

Esse método muda o nível, definido na propriedade Level, de um item do tipo
```
TOutlineNode.
```

Declaração
```
procedure ChangeLevelBy(Value: TChangeRange);
```

Componentes aos quais se aplica:
```
TOutlineNode
```

CHECKACTIVE

Descrição

Esse método verifica se uma conexão a um banco de dados do Interbase está ativa.

Declaração
```
procedure CheckActive;
```

Componentes aos quais se aplica:
```
TIBDatabase
```

CHECKBROWSEMODE

Descrição

Esse método verifica se o banco de dados está aberto e se possui alterações pendentes. Se sua propriedade State é igual a dsEdit, dsInsert ou dsSetKey, o método Post é chamado para concluir as operações pendentes. Se o banco de dados estiver fechado, gera um erro de exceção do tipo EDataBaseError.

Declaração
```
procedure CheckBrowseMode;
```

Componentes aos quais se aplica:
```
TADODataset, TADOTable, TADOQuery, TADOStoredProc,TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery e TStoredProc, TIBSQLClientDataset, TSimpleDataset, TSQLDataset,
TSQLTable e TSQLQuery
```

CHECKCLOSED

Descrição

Esse método gera uma exceção se a query definida pelo componente não estiver inativa.

Declaração
```
procedure CheckClosed;
```

Componentes aos quais se aplica:
TIBSQL

CheckDatabaseInList

Descrição

Esse método verifica se só existem componentes TIBDatabase na lista de objetos manipulada pelo componente.

Declaração
```
procedure CheckDatabasesInList;
```

Componentes aos quais se aplica:
TIBTransaction

CheckDatabasename

Descrição

Esse método verifica se a propriedade DatabaseName do componente está em branco.

Declaração
```
procedure CheckDatabaseName;
```

Componentes aos quais se aplica:
TIBDatabase

CheckInactive

Descrição

Esse método verifica se uma conexão a um banco de dados do Interbase está inativa.

Declaração
```
procedure CheckInctive;
```

Componentes aos quais se aplica:
TIBDatabase

CheckInTransaction

Descrição

Esse método verifica se a transação está ativa e se só existem componentes TIBDatabase na lista de objetos manipulada pelo componente.

Declaração
```
procedure CheckInTransaction;
```

Componentes aos quais se aplica:
TIBTransaction

CheckNotInTransaction

Descrição

Esse método verifica se a transação não está ativa e se não existem componentes TIBDatabase na lista de objetos manipulada pelo componente.

Declaração
```
procedure CheckNotInTransaction;
```

Componentes aos quais se aplica:
TIBTransaction

CheckOpen

Descrição

Esse método gera uma exceção se a query definida pelo componente não estiver ativa.

Declaração
```
procedure CheckOpen;
```

Componentes aos quais se aplica:
TIBSQL

CheckValidStatement

Descrição

Esse método gera uma exceção se a query definida pelo componente não possuir uma declaração SQL válida.

Declaração
```
procedure CheckValidStatement;
```

Componentes aos quais se aplica:
TIBSQL

ClassInfo

Descrição

Esse método retorna um ponteiro para a tabela RTTI (run time type information) que contém informações sobre o objeto corrente.

Declaração
```
class function ClassInfo: Pointer;
```

Componentes aos quais se aplica:
Todos os objetos.

ClassName

Descrição

Esse método retorna em uma string o nome da classe ou objeto do componente.

Declaração
```
class function ClassName: string;
```

Exemplo

O trecho de código a seguir faz com que um componente Label1 do tipo TLabel exiba o nome da sua classe quando o usuário dá um clique com o mouse sobre um componente Button1 do tipo TButton:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
         Label1.Caption := Label1.ClassName;
end;
```

Observação: Será exibido TLabel.

Componentes aos quais se aplica:

Todos os objetos e componentes.

ClassNameIs

Descrição

Esse método define se a string passada no parâmetro Name é igual ao nome da classe.

Declaração

```
class function ClassNameIs(const Name: string): Boolean;
```

Componentes aos quais se aplica:

Todos os objetos.

ClassParent

Descrição

Esse método retorna em uma variável TClass a classe ou objeto ancestral do qual o componente é derivado.

Declaração

```
class function ClassParent: TClass;
```

Exemplo

O trecho de código a seguir faz com que um componente Label1 do tipo TLabel exiba o nome da classe da qual é derivado quando o usuário dá um clique com o mouse sobre um componente Button1 do tipo TButton:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    classe : TClass;
begin
     classe := Label1.ClassParent;
         Label1.Caption := classe.ClassName;
end;
```

Observação: Será exibido TCustomLabel.

Componentes aos quais se aplica:

Todos os objetos e componentes.

ClassType

Descrição

Esse método retorna a classe ou objeto do componente.

Declaração

```
function ClassType: TClass;
```

Exemplo

O trecho de código a seguir faz com que um componente Label1 do tipo TLabel exiba o nome da sua classe quando o usuário dá um clique com o mouse sobre um componente Button1 do tipo TButton:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    classe : TClass;
begin
     classe := Label1.ClassType;
         Label1.Caption := classe.ClassName;
end;
```

Observação: Será exibido TLabel.

Componentes aos quais se aplica:

Todos os objetos e componentes.

CleanUp

Descrição

Esse método libera a memória ocupada pelas páginas de um relatório após o término do trabalho de impressão de um relatório e atribui o valor mbReady à propriedade Status do componente.

Declaração

```
procedure CleanUp;
```

Componentes aos quais se aplica:

```
TQRPrinter
```

CleanupInstance

Descrição

Esse método finaliza qualquer string ou registro definido na classe e é chamado pelo método FreeInstance.

Declaração

```
procedure CleanupInstance;
```

Componentes aos quais se aplica:

Todos os objetos.

Clear

Descrição

Para componentes dos tipos TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField, TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField, TStringField, TTimeField, TVarBytesField e TWordField, esse método atribui NULL ao valor do campo correspondente.

Para componentes dos tipos TFieldDefs e TIndexDefs, esse método libera todas as entradas da sua propriedade Items.

Para objetos do tipo TParam, esse método tem seu valor como NULL.

Para objetos do tipo TParams, esse método deleta toda a informação da sua propriedade Items.

Para objetos dos tipos TClipBoard, esse método remove o conteúdo da área de transferência. Isso ocorre automaticamente cada vez que um dado é colocado na área de transferência por meio das operações Cut e Copy.

Para objetos dos tipos TList, TStringList, TStrings, TComboBox, TDBComboBox, TDBListBox, TDBMemo, TDirectoryListBox, TDriveComboBox, TFileListBox, TFilterComboBox, TListBox, TMemo e TOutline, esse método deleta todas as entradas na propriedade Items.

Para controles dos tipos TDBEdit, TEdit e TMaskEdit, esse método remove o texto da propriedade Text do controle.

Para objetos da classe TChartSeries, remove todos os pontos da série.

Para componentes das classes TIWApplet, TIWButton, TIWCheckbox, TIWCombobox, TIWControl, TIWDBCheckbox, TIWDBEdit, TIWDBFile, TIWDBGrid, TIWDBImage, TIWDBListbox, TIWDBLookupCombobox, TIWDBLookupListbox, TIWDBMemo, TIWDBNavigator, TIWDBText, TIWEdit, TIWForm, TIWGrid, TIWImage, TIWImageFile, TIWLabel, TIWLink, TIWList, TIWListbox, TIWMemo, TIWRectangle, TIWTimer, TIWTreeview e TIWURL, limpa a renderização atual antes de uma nova renderização.

Declaração

```
procedure Clear;
```

Exemplo

O trecho de código abaixo remove todos os itens de um componente ListBox1 do tipo

```
TListBox:
ListBox1.Clear;
```

Componentes aos quais se aplica:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TChartSeries,
TCurrencyField, TDateField, TDateTimeField, TFloatField, TGraphicField, TIntegerField,
TMemoField, TSmallintField, TStringField, TTimeField, TVarBytesField, TWordField, TFieldDefs,
TIndexDefs, TParam, TParams, TClipBoard, TList, TStringList, TStrings, TComboBox, TDBComboBox,
TDBListBox, TDBMemo, TDirectoryListBox, TDriveComboBox, TFileListBox, TFilterComboBox,
TListBox, TMemo, TOutline, TDBEdit, TEdit e TMaskEdit
```

CLEARFIELDS

Descrição

Esse método atribui o valor default a todos os campos do registro corrente em um banco de dados se sua propriedade State é igual a dsEdit; se não, gera um erro de exceção do tipo EDataBaseError.

Declaração

```
procedure ClearFields;
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc,TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery e TStoredProc
```

CLEARSELECTION

Descrição

Esse método deleta o texto selecionado em um componente.

Declaração

```
procedure ClearSelection;
```

Exemplo

O trecho de código abaixo deleta o item selecionado em um componente Memo1 do tipo

```
TMemo:
Memo1.ClearSelection;
```

Componentes aos quais se aplica:

```
TDBEdit, TDBMemo, TEdit, TMaskEdit e TMemo
```

CLICK

Descrição

Esse método simula um clique do mouse sobre um componente, forçando a execução do código associado ao evento OnClick.

Declaração

Para componentes do tipo TDBNavigator:

```
procedure Click(Button: TNavigateBtn);
```

Para componentes dos tipos TBitBtn, TButton, TMenuItem e TSpeedButton:

```
procedure Click;
```

Exemplo

O trecho de código a seguir faz com que o efeito de clicar sobre um botão chamado Button1 seja o mesmo de clicar sobre um botão chamado Button2.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     Button2.Click;
end;
```

Componentes aos quais se aplica:

Componentes TBitBtn, TButton, TDBNavigator, TMenuItem e TSpeedButton

ClientToScreen

Descrição

Esse método faz a transformação das coordenadas de um ponto do sistema de coordenadas da área-cliente para o sistema de coordenadas da tela.

Declaração

```
function ClientToScreen(Point: TPoint): TPoint;
```

Exemplo

O trecho de código que se segue define P e Q como variáveis do tipo TPoint no evento OnMouseDown de um formulário. Ao ponto P são atribuídas as coordenadas do ponto em que o botão do mouse foi pressionado (sistema de coordenadas da área-cliente – o formulário) e esses valores são armazenados no ponto Q após a transformação para o sistema de coordenadas da tela.

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
 Shift: TShiftState; X, Y: Integer);
var
 P, Q: TPoint;
begin
 P.X:= X;
 P.Y:= Y;
 Q:= ClientToScreen(P);
end;
```

Observação: Note que esse método é uma função e, conseqüentemente, o seu valor de retorno pode ser atribuído a uma variável.

Componentes aos quais se aplica:

Todos os controles.

Close

Descrição

Para componentes do tipo TForm, esse método executa o código correspondente ao evento OnClose após tentar fechar um formulário chamando o método CloseQuery cujo valor booleano de retorno (True ou False) determina se o formulário pode realmente ser fechado.

Para componentes do tipo TMediaPlayer, esse método fecha o dispositivo multimídia atualmente aberto.

Para objetos do tipo TClipBoard, fecha o objeto.

Para componentes dos tipos TClientDataSet, TTable, TQuery e TStoredProc, fecha o banco de dados associado e o deixa inativo.

Para componentes dos tipos TADOConnection, TIBDatabase, TDatabase, fecha o banco de dados e todos os componentes dos tipos TTable, TQuery e TStoredProc a ele relacionados.

Para componentes do tipo TRvProject, fecha o projeto de relatório representado pelo componente.

Declaração

```
procedure Close;
```

Exemplo

O trecho de código a seguir faz com que o método Close de um formulário denominado Form1 seja chamado ao se clicar sobre um botão denominado Button1 do tipo TButton.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     Form1.Close;
end;
```

Componentes aos quais se aplica:

```
TADOConnection, TADODataset, TADOTable, TADOQuery, TADOStoredProc, TClipBoard, TForm,
TIBDatabase, IBDataset, TIBTable, TIBQuery, TIBStoredProc, TMediaPlayer, TQuery, TRvProject,
TStoredProc e TTable
```

CloseDatabase

Descrição

O método CloseDatabase é usado para fechar um componente do tipo TDatabase usado pela aplicação, definido no parâmetro Database.

Declaração

```
procedure CloseDatabase(Database: TDatabase);
```

Exemplo

O trecho de código a seguir fecha um banco de dados chamado Dados:

```
Session.CloseDatabase('DADOS');
```

Componentes aos quais se aplica:

```
TSession
```

CloseDataSets

Descrição

Fecha todos os componentes dos tipos derivados de TDataset a ele relacionados, sem fechar o banco de dados.

Declaração

```
procedure CloseDataSets;
```

Componentes aos quais se aplica:

```
TDatabase, TADOConnection, TSQLConnection e TIBDatabase
```

CloseDialog

Descrição

Esse método fecha a caixa de diálogo.

Declaração

```
procedure CloseDialog;
```

Exemplo

O trecho de código a seguir fecha a caixa de diálogo FindDialog1 do tipo TFindDialog:

```
FindDialog1.CloseDialog;
```

Componentes aos quais se aplica:
TFindDialog e TReplaceDialog

CLOSELINK

Descrição

Esse método encerra uma conversação DDE.

Declaração
```
function CloseLink;
```

Exemplo

O trecho de código a seguir encerra uma conversação DDE.
```
DDEClientConv1.CloseLink;
```

Componentes aos quais se aplica:
TDDEClientConv

CLOSEQUERY

Descrição

Esse método executa o código associado ao evento OnCloseQuery de um formulário e seu valor de retorno determina se um formulário poderá ou não ser fechado. Se o formulário for do tipo MDI, chamará o método OnCloseQuery de todas as suas janelas-filhas, e se uma delas retornar False, este também retornará False.

Declaração
```
function CloseQuery: Boolean;
```

Exemplo

O trecho de código a seguir faz com que uma mensagem de confirmação seja exibida quando o usuário tentar fechar um formulário:
```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
var
      BotResp: Word;
begin
  BotResp:= MessageDlg('Fechar o formulário?', mtInformation,[mbOk, mbNo],0);
      if BotResp = mrOk then
   CanClose:= True
      else
   CanClose:= False;
end;
```

Componentes aos quais se aplica:
TForm

CLOSEUP

Descrição

Esse método fecha a lista drop-down aberta no componente.

Declaração
```
procedure CloseUp;
```

Exemplo

O trecho de código a seguir executa o método CloseUp em um componente DBLookupCombo1 do tipo TDBLookupCombo.
```
DBLookupCombo1.CloseUp;
```

Componentes aos quais se aplica:
TDBLookupCombo e TDBLookupComboBox

Collapse

Descrição

Esse método fecha um item do tipo TOutlineNode, atribuindo o valor False à sua propriedade Expanded.

Declaração
```
procedure Collapse;
```

Componentes aos quais se aplica:
TOutlineNode

ColorToRGBString

Descrição

Esse método converte uma variável do tipo TColor passada como parâmetro em uma string que define a representação no formato RGB desta mesma cor.

Declaração
```
class function ColorToRGBString(AColor: TColor): string;
```

Componentes aos quais se aplica:
TIWApplet, TIWButton, TIWCheckbox,

Commit

Descrição

Esse método finaliza as transações pendentes com o banco de dados.

Declaração
```
procedure Commit;
```

Componentes aos quais se aplica:
TDatabase, TIBTransaction e TSQLConnection.

CommitRetaining

Descrição

Esse método finaliza as transações pendentes com o banco de dados e mantém a transação ativa.

Declaração
```
procedure CommitRetaining;
```

Componentes aos quais se aplica:
TDatabase, TIBTransaction

CommitTrans

Descrição

Esse método finaliza as transações pendentes com o banco de dados.

Declaração
```
procedure CommitTrans;
```

Componentes aos quais se aplica:
TADOConnection

CommitUpdates

Descrição

O método CommitUpdates é usado para gravar permanentemente as modificações efetuadas nos valores de uma tabela armazenados no seu cache.

Declaração

```
procedure CommitUpdates;
```

Componentes aos quais se aplica:

```
TTable
```

Connect

Descrição

Esse método inicializa a conexão desejada.

Declaração

```
procedure Connect; override;
```

Componentes aos quais se aplica:

```
TNMEcho, TNMFTP, TNMNNTP, TNMPOP3, TNMSMTP, TpowerSock
```

ContainsControl

Descrição

Esse método verifica se um controle contém ou não um outro controle.

Declaração

```
function ContainsControl(Control: TControl): Boolean;
```

Exemplo

O trecho de código a seguir verifica se um componente chamado Panel1 do tipo TPanel contém um componente Edit1 do tipo TEdit.

```
if Panel1.ContainsControl(Edit1) then ShowMessage('Contém') else ShowMessage('Não Contém');
```

Componentes aos quais se aplica:

```
Todos os controles.
```

Content

Descrição

Esse método retorna em uma string a página HTML gerada pelo componente.

Declaração

```
function Content: string; override;
```

Componentes aos quais se aplica:

```
TMidasPageProducer, TpageProducer, TdataseTTableProducer, TQueryTableproducer,
TDatasetPageProducer
```

ControlAtPos

Descrição

Esse método retorna o controle localizado na posição definida pelo parâmetro Pos.

Declaração

```
function ControlAtPos(Pos: TPoint; AllowDisabled: Boolean): TControl;
```

Componentes aos quais se aplica:

Todos os controles.

CopyParams

Descrição

Esse método copia as informações do procedimento armazenado no servidor no parâmetro Value.

Declaração

```
procedure CopyParams(Value: TParams);
```

Componentes aos quais se aplica:

```
TStoredProc
```

CopyRect

Descrição

Esse método copia uma área retangular (definida no parâmetro Source) de um canvas (definido no parâmetro Canvas) em uma área retangular (definida no parâmetro Dest) do canvas corrente.

Declaração

```
procedure CopyRect(Dest: TRect; Canvas: TCanvas; Source: TRect);
```

Componentes aos quais se aplica:

```
TCanvas
```

CopyToClipboard

Descrição

Para componentes do tipo TDDEServerConv, esse método transfere o texto definido na propriedade Lines ou na propriedade Text para o clipboard.

Para componentes dos tipos TDBEdit, TDBMemo, TEdit, TMaskEdit e TMemo, esse método copia o texto selecionado no controle para o clipboard, substituindo qualquer texto já existente.

Declaração

```
procedure CopyToClipboard;
```

Exemplo

O trecho de código a seguir copia o texto selecionado em um controle Edit1 do tipo TEdit para o clipboard:

```
Edit1.CopyToClipboard;
```

Componentes aos quais se aplica:

```
TDBEdit, TDBMemo, TEdit, TMaskEdit e TMemo
```

CorbaObject

Descrição

Esse método retorna um ponteiro para o objeto CORBA associado a essa interface.

Declaração

```
function CorbaObject: PCorbaObject; stdcall;
```

Interfaces às quais se aplica:

```
ICorbaObj
```

Create

Descrição

Esse método faz com que um objeto ou componente seja criado.

Para objetos do tipo TIniFile, aloca a memória necessária para a criação do objeto e passa como argumento um arquivo .INI. Caso o arquivo não esteja no diretório do Windows, deve-se passar também o seu path completo.

Para objetos do tipo TOutline, cria um novo nó para o objeto TOutline passado pelo parâmetro AOwner.

Para objetos do tipo TOutline, cria um novo nó para o objeto TOutline passado pelo parâmetro AOwner.

Para objetos do tipo TControlScrollBar, cria uma nova barra de rolagem no controle definido pelo parâmetro AControl. O tipo da barra de rolagem é definido no parâmetro AKind, que pode ser sbHorizontal ou sbVertical.

Para objetos do tipo TIndexDef, cria um novo objeto com os parâmetros Name, Fields e Options e o adiciona à propriedade Items de um objeto definido pelo parâmetro Owner.

Para objetos do tipo TBlobStream, cria e faz a ligação de um objeto dos tipos TBlobField, TBytesField ou TVarBytesField. O parâmetro Mode pode ser igual a bmRead, para acessar os dados existentes; bmWrite, para substituir o valor do campo; e bmReadWrite, para alterar um valor existente.

Para os demais componentes, aloca memória para o componente e inicializa os dados necessários.

Para os demais objetos, aloca memória para o objeto e inicializa os dados necessários.

Declaração

A declaração do método dependerá do objeto a que se refere.

Para objetos do tipo TIniFile:

```
constructor Create(const FileName: string);
```

Para objetos do tipo TOutline:

```
constructor Create(AOwner: TCustomOutline);
```

Para objetos do tipo TControlScrollBar:

```
constructor Create (AControl: TScrollingWinControl; AKind: TScrollBarKind);
```

Para objetos do tipo TIndexDef:

```
constructor Create(Owner: TIndexDefs; const Name, Fields: string; Options: TIndexOptions);
```

Para objetos do tipo TBlobStream:

```
constructor Create(Field: TBlobField; Mode: TBlobStreamMode);
```

Para os demais componentes:

```
constructor Create(AOwner: TComponent);
```

Para os demais objetos:

```
constructor Create;
```

Exemplo

O trecho de código a seguir cria um botão chamado Button1 do tipo TButton, pertencente

```
a um formulário chamado Form1.
var
    Button1 : TButton;
begin
    Button1 := TButton.Create(Form1);
end;
```

Componentes aos quais se aplica:

Todas as classes, componentes e objetos.

CreateDatabase

Descrição

Esse método cria um novo banco de dados, configurando-o de acordo com os valores armazenados na sua propriedade Params.

Declaração

```
procedure CreateDatabase;
```

Componentes aos quais se aplica:

```
TIBDatabase
```

CreateField

Descrição

Esse método cria um novo objeto do tipo TField, pertencente ao componente definido pelo parâmetro Owner.

Declaração

```
function CreateField(Owner: TComponent): TField;
```

Componentes aos quais se aplica:

```
TFieldDef
```

CreateForm

Descrição

Esse método cria um novo formulário para a aplicação, cujo tipo é especificado no parâmetro FormClass, e o atribui à variável definida pelo parâmetro Reference.

Declaração

```
procedure CreateForm(FormClass: TFormClass; var Reference);
```

Componentes aos quais se aplica:

```
TApplication
```

CreateFmt

Descrição

Esse é outro método construtor de classes derivadas de Exception e recebe como parâmetros uma string, que será mostrada na caixa de diálogo exibida quando a exceção ocorrer durante a execução do programa, e uma array contendo códigos de formatação para essa string.

Declaração

```
constructor CreateFmt (const Msg: string; const Args: array of const)
```

Componentes aos quais se aplica:

```
EAbort, EAbstractError, EAccessViolation, EArrayError, EAssertinFailed, EBitsError,
EBrokerException, ECacheError, EClassNotFound, ECommomCalendarError, EComponentError,
EControlC, EConvertError, EDataBaseError, EDateTimeError, EDBClient, EDBEditError,
EDBEngineError, EDimensionMapError, EDimIndexError, EDivByZero, EDSWriter, EExternal,
EExternalException, EFCreateError, EFilerError, EFOpenError, EHeapException, EInOutError,
EInterpreterError, EIntError, EIntCastError, EIntOverFlow, EInvalidArgument, EInvalidCast,
EInvalidGraphic, EInvalidGraphicOperation, EInvalidGridOperation, EInvalidImage, EInvalidOp,
EInvalidOperation, EInvalidPointer, EListError, ELowCapacityError, EMathError,
EMCIDeviceError, EMenuError, EMonthCalError, ENoResultError, EOLECtrlError, EOLEError,
EOLEException, EOLESysError, EOutLineError, EOutOfMemory, EOutOfResources, EOverflow,
EPackageError, EParserError, EPrinter, EPrivilege, EPropertyError, EPropReadOnly,
```

```
EPropWriteOnly, ERangeError, ERadError, EReconcileError, ERegistryError, EResNotFound,
ESocketConnectionError, ESocketError, EStackOverflow, EStreamError, EStringListError, EThread,
ETreeViewError, EUnderFlow, EUnSupportedTypeError, EUpdateError, EVariantError, EWin32Error,
EWriteError, Exception, EZeroDivide
```

CreateFmtHelp

Descrição

Esse é outro método construtor de classes derivadas de Exception e recebe como parâmetros uma string, que será mostrada na caixa de diálogo exibida quando a exceção ocorrer durante a execução do programa; uma array, contendo códigos de formatação para essa string; e um inteiro, identificando o tópico do arquivo de Help On-line a ser associado à classe.

Declaração

```
constructor CreateFmtHelp (const Msg: string; const Args: array of const; AHelpContext: Integer);
```

Componentes aos quais se aplica:

```
EAbort, EAbstractError, EAccessViolation, EArrayError, EAssertinFailed, EBitsError,
EBrokerException, ECacheError, EClassNotFound, ECommomCalendarError, EComponentError,
EControlC, EConvertError, EDataBaseError, EDateTimeError, EDBClient, EDBEditError,
EDBEngineError, EDimensionMapError, EDimIndexError, EDivByZero, EDSWriter, EExternal,
EExternalException, EFCreateError, EFilerError, EFOpenError, EHeapException, EInOutError,
EInterpreterError, EIntError, EIntCastError, EIntOverFlow, EInvalidArgument, EInvalidCast,
EInvalidGraphic, EInvalidGraphicOperation, EInvalidGridOperation, EInvalidImage, EInvalidOp,
EInvalidOperation, EInvalidPointer, EListError, ELowCapacityError, EMathError,
EMCIDeviceError, EMenuError, EMonthCalError, ENoResultError, EOLECtrlError, EOLEError,
EOLEException, EOLESysError, EOutLineError, EOutOfMemory, EOutOfResources, EOverflow,
EPackageError, EParserError, EPrinter, EPrivilege, EPropertyError, EPropReadOnly,
EPropWriteOnly, ERangeError, ERadError, EReconcileError, ERegistryError, EResNotFound,
ESocketConnectionError, ESocketError, EStackOverflow, EStreamError, EStringListError, EThread,
ETreeViewError, EUnderFlow, EUnSupportedTypeError, EUpdateError, EVariantError, EWin32Error,
EWriteError, Exception, EZeroDivide
```

CreateHelp

Descrição

Esse é outro método construtor de classes derivadas de Exception e recebe como parâmetros uma string, que será mostrada na caixa de diálogo exibida quando a exceção ocorrer durante a execução do programa, e um inteiro, identificando o tópico do arquivo

de Help On-line a ser associado à classe.

Declaração

```
constructor CreateHelp (const Msg: string; AHelpContext: Integer);
```

Componentes aos quais se aplica:

```
EAbort, EAbstractError, EAccessViolation, EArrayError, EAssertinFailed, EBitsError,
EBrokerException, ECacheError, EClassNotFound, ECommomCalendarError, EComponentError,
EControlC, EConvertError, EDataBaseError, EDateTimeError, EDBClient, EDBEditError,
EDBEngineError, EDimensionMapError, EDimIndexError, EDivByZero, EDSWriter, EExternal,
EExternalException, EFCreateError, EFilerError, EFOpenError, EHeapException, EInOutError,
EInterpreterError, EIntError, EIntCastError, EIntOverFlow, EInvalidArgument, EInvalidCast,
EInvalidGraphic, EInvalidGraphicOperation, EInvalidGridOperation, EInvalidImage, EInvalidOp,
EInvalidOperation, EInvalidPointer, EListError, ELowCapacityError, EMathError,
EMCIDeviceError, EMenuError, EMonthCalError, ENoResultError, EOLECtrlError, EOLEError,
EOLEException, EOLESysError, EOutLineError, EOutOfMemory, EOutOfResources, EOverflow,
EPackageError, EParserError, EPrinter, EPrivilege, EPropertyError, EPropReadOnly,
EPropWriteOnly, ERangeError, ERadError, EReconcileError, ERegistryError, EResNotFound,
ESocketConnectionError, ESocketError, EStackOverflow, EStreamError, EStringListError, EThread,
ETreeViewError, EUnderFlow, EUnSupportedTypeError, EUpdateError, EVariantError, EWin32Error,
EWriteError, Exception, EzeroDivide
```

CreateIndexFile

Descrição

Esse método cria um arquivo de índices para tabelas no formato dBASE.

Declaração

```
procedure CloseIndexFile(const IndexFileName: string);
```

Componentes aos quais se aplica:

```
TTable
```

CreateNew

Descrição

Esse método cria uma nova instância do formulário corrente.

Declaração

```
constructor CreateNew(AOwner: TComponent);
```

Componentes aos quais se aplica:

```
TForm
```

CreateRes

Descrição

Esse é outro método construtor de classes derivadas de Exception e recebe como parâmetro um inteiro que identifica uma string armazenada no arquivo de recursos do seu aplicativo.

Declaração

```
constructor CreateRes (Ident: Integer; Dummy: Extended = 0);
```

Componentes aos quais se aplica:

```
EAbort, EAbstractError, EAccessViolation, EArrayError, EAssertinFailed, EBitsError,
EBrokerException, ECacheError, EClassNotFound, ECommomCalendarError, EComponentError,
EControlC, EConvertError, EDataBaseError, EDateTimeError, EDBClient, EDBEditError,
EDBEngineError, EDimensionMapError, EDimIndexError, EDivByZero, EDSWriter, EExternal,
EExternalException, EFCreateError, EFilerError, EFOpenError, EHeapException, EInOutError,
EInterpreterError, EIntError, EIntCastError, EIntOverFlow, EInvalidArgument, EInvalidCast,
EInvalidGraphic, EInvalidGraphicOperation, EInvalidGridOperation, EInvalidImage, EInvalidOp,
EInvalidOperation, EInvalidPointer, EListError, ELowCapacityError, EMathError,
EMCIDeviceError, EMenuError, EMonthCalError, ENoResultError, EOLECtrlError, EOLEError,
EOLEException, EOLESysError, EOutLineError, EOutOfMemory, EOutOfResources, EOverflow,
EPackageError, EParserError, EPrinter, EPrivilege, EPropertyError, EPropReadOnly,
EPropWriteOnly, ERangeError, ERadError, EReconcileError, ERegistryError, EResNotFound,
ESocketConnectionError, ESocketError, EStackOverflow, EStreamError, EStringListError, EThread,
ETreeViewError, EUnderFlow, EUnSupportedTypeError, EUpdateError, EVariantError, EWin32Error,
EWriteError, Exception, EZeroDivide
```

CreateResFmt

Descrição

Esse é outro método construtor de classes derivadas de Exception e recebe como parâmetros um inteiro, que identifica uma string armazenada no arquivo de recursos do seu aplicativo, e uma array, contendo códigos de formatação para essa string.

Declaração

```
constructor CreateResFmt(Ident: Integer; const Args: array of const);
```

Componentes aos quais se aplica:

```
EAbort, EAbstractError, EAccessViolation, EArrayError, EAssertinFailed, EBitsError,
EBrokerException, ECacheError, EClassNotFound, ECommomCalendarError, EComponentError,
```

```
EControlC, EConvertError, EDataBaseError, EDateTimeError, EDBClient, EDBEditError,
EDBEngineError, EDimensionMapError, EDimIndexError, EDivByZero, EDSWriter, EExternal,
EExternalException, EFCreateError, EFilerError, EFOpenError, EHeapException, EInOutError,
EInterpreterError, EIntError, EIntCastError, EIntOverFlow, EInvalidArgument, EInvalidCast,
EInvalidGraphic, EInvalidGraphicOperation, EInvalidGridOperation, EInvalidImage, EInvalidOp,
EInvalidOperation, EInvalidPointer, EListError, ELowCapacityError, EMathError,
EMCIDeviceError, EMenuError, EMonthCalError, ENoResultError, EOLECtrlError, EOLEError,
EOLEException, EOLESysError, EOutLineError, EOutOfMemory, EOutOfResources, EOverflow,
EPackageError, EParserError, EPrinter, EPrivilege, EPropertyError, EPropReadOnly,
EPropWriteOnly, ERangeError, ERadError, EReconcileError, ERegistryError, EResNotFound,
ESocketConnectionError, ESocketError, EStackOverflow, EStreamError, EStringListError, EThread,
ETreeViewError, EUnderFlow, EUnSupportedTypeError, EUpdateError, EVariantError, EWin32Error,
EWriteError, Exception, EZeroDivide
```

CreateResFmtHelp

Descrição

Esse é outro método construtor de classes derivadas de Exception e recebe como parâmetros um inteiro, que identifica uma string armazenada no arquivo de recursos do seu aplicativo; uma array, contendo códigos de formatação para essa string; e um valor inteiro, que identifica o código do arquivo de Help associado a essa classe.

Declaração

```
constructor CreateResFmtHelp(Ident: Integer; const Args: array of const; AHelpContext:
Integer);
```

Componentes aos quais se aplica:

```
EAbort, EAbstractError, EAccessViolation, EArrayError, EAssertinFailed, EBitsError,
EBrokerException, ECacheError, EClassNotFound, ECommomCalendarError, EComponentError,
EControlC, EConvertError, EDataBaseError, EDateTimeError, EDBClient, EDBEditError,
EDBEngineError, EDimensionMapError, EDimIndexError, EDivByZero, EDSWriter, EExternal,
EExternalException, EFCreateError, EFilerError, EFOpenError, EHeapException, EInOutError,
EInterpreterError, EIntError, EIntCastError, EIntOverFlow, EInvalidArgument, EInvalidCast,
EInvalidGraphic, EInvalidGraphicOperation, EInvalidGridOperation, EInvalidImage, EInvalidOp,
EInvalidOperation, EInvalidPointer, EListError, ELowCapacityError, EMathError,
EMCIDeviceError, EMenuError, EMonthCalError, ENoResultError, EOLECtrlError, EOLEError,
EOLEException, EOLESysError, EOutLineError, EOutOfMemory, EOutOfResources, EOverflow,
EPackageError, EParserError, EPrinter, EPrivilege, EPropertyError, EPropReadOnly,
EPropWriteOnly, ERangeError, ERadError, EReconcileError, ERegistryError, EResNotFound,
ESocketConnectionError, ESocketError, EStackOverflow, EStreamError, EStringListError, EThread,
ETreeViewError, EUnderFlow, EUnSupportedTypeError, EUpdateError, EVariantError, EWin32Error,
EWriteError, Exception, EZeroDivide
```

CreateSize

Descrição

Esse método cria um objeto do tipo TImageList cujo tamanho é definido pelos parâmetros Height (altura) e Width (largura).

Declaração

```
constructor CreateSize(AWidth, AHeight: Integer);
```

Componentes aos quais se aplica:

```
TImageList
```

CreateTable

Descrição

Esse método cria uma nova tabela para o banco de dados associado.

Declaração

```
procedure CreateTable;
```

Componentes aos quais se aplica:

TIBTable, TTable

CursorPosChanged

Descrição

Esse método só é utilizado quando a propriedade Handle é usada para obter acesso direto à API do Borland Database Engine ou outro mecanismo de acesso para notificar o banco de dados sobre a mudança da posição do cursor.

Declaração

```
procedure CursorPosChanged;
```

Componentes aos quais se aplica:

TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery, TIBStoredProc, TTable, TQuery, TStoredProc, TSQLDataset, TSQLTable, TSQLQuery e TSQLStoredProc

CutToClipboard

Descrição

Esse método copia o texto selecionado no controle para o clipboard, deletando-o e substituindo qualquer texto já existente no clipboard.

Declaração

```
procedure CutToClipboard;
```

Exemplo

O trecho de código a seguir copia o texto selecionado em um controle Edit1 do tipo TEdit para o clipboard.

```
Edit1.CutToClipboard;
```

Componentes aos quais se aplica:

TDBEdit, TDBImage, TDBMemo, TEdit, TMaskEdit e TMemo

Delete

Descrição

Para componentes derivados da classe TDataset, esse método só é usado para deletar um registro de um banco de dados (o registro anterior passa a ser o registro corrente, exceto se o registro deletado for o último do banco de dados – nesse caso o registro anterior passa a ser o corrente).

Para objetos dos tipos TImageList, TList, TStringList e TStrings e componentes do tipo TMenuItem, deleta o item especificado no parâmetro Index.

Declaração

Para componentes dos tipos TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery, TIBStoredProc, TTable, TQuery, TStoredProc, TSimpleDataset, TSQLDataset, TSQLTable, TSQLQuery e TSQLStoredProc.

```
procedure Delete;
```

Para objetos dos tipos TImageList, TList, TStringList e TStrings e componentes do tipo TMenuItem:

```
procedure Delete(Index: Integer);
```

Componentes aos quais se aplica:
```
TImageList, TList, TStringList, TStrings, TMenuItem, TADODataset, TADOTable, TADOQuery,
TADOStoredProc, TIBDataset, TIBTable, TIBQuery, TIBStoredProc, TTable, TQuery, TStoredProc,
TSimpleDataset, TSQLDataset, TSQLTable, TSQLQuery e TSQLStoredProc
```

DeleteIndex

Descrição

Esse método deleta um índice secundário (definido no parâmetro Name) em uma tabela.

Declaração
```
procedure DeleteIndex(const Name: string);
```

Componentes aos quais se aplica:
```
TIBTable, TTable
```

DeleteTable

Descrição

Esse método deleta uma tabela de um banco de dados.

Declaração
```
procedure DeleteTable;
```

Componentes aos quais se aplica:
```
TIBTable, TTable
```

DescriptionAvailable

Descrição

Esse método verifica se as informações do procedimento armazenado no servidor estão disponíveis.

Declaração
```
function DescriptionsAvailable: Boolean;
```

Componentes aos quais se aplica:
```
TStoredProc
```

DesignReport

Descrição

Esse método abre o arquivo de projeto de relatório para edição.

Declaração
```
procedure DesignReport(ReportName: string);
```

Componentes aos quais se aplica:
```
TRvProject
```

Destroy

Descrição

Esse método destrói um componente ou objeto, liberando a memória a ele alocada.

Declaração
```
destructor Destroy;
```

Exemplo

A linha de código a seguir mostra um botão Button1 do tipo TButton executando o seu método Destroy.
```
Button1.Destroy;
```

Componentes aos quais se aplica:

Todos os objetos, controles e componentes.

DestroyComponents

Descrição

Esse método destrói os componentes-filhos do componente corrente.

Declaração

```
procedure DestroyComponents;
```

Componentes aos quais se aplica:

Todos os componentes.

Destroying

Descrição

Esse método altera o valor da propriedade ComponentState dos componentes-filhos do componente corrente e os destrói.

Declaração

```
procedure Destroying;
```

Componentes aos quais se aplica:

Todos os componentes.

DisableAlign

Descrição

Esse método desabilita o realinhamento automático dos controles-filhos do controle corrente.

Declaração

```
procedure DisableAlign;
```

Componentes aos quais se aplica:

Todos os controles.

DisableControls

Descrição

Esse método desconecta temporariamente o banco de dados de todos os componentes a ele associados (do tipo TDataSource).

Declaração

```
procedure DisableControls;
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TSimpleDataset, TSQLDataset, TSQLTable, TSQLQuery,
TSQLStoredProc, TClientDataset, TSimpleDataset e TIBClientDataset
```

DisConnect

Descrição

Esse método encerra a conexão atual.

Declaração

```
procedure DisConnect; override;
```

Componentes aos quais se aplica:

```
TNMEcho, TNMFTP, TNMNNTP, TNMPOP3, TNMSMTP, TPowerSock
```

Dispatch

Descrição

Esse método chama os métodos de manipulação de mensagens, dependendo do valor passado no parâmetro Message.

Declaração

```
procedure Dispatch(var Message);
```

Componentes aos quais se aplica:

Todos os objetos.

DoHint

Descrição

Esse método executa o procedimento associado ao evento OnHint do objeto (se este estiver definido), passando como parâmetro uma string representando seu hint.

Declaração

```
function DoHint(var HintStr: string): Boolean;
```

Componentes aos quais se aplica:

```
TAction
```

DoTerminate

Descrição

Esse método dispara o procedimento associado ao evento OnTerminate de uma thread, sem, no entanto, finalizá-la.

Declaração

```
procedure DoTerminate; virtual;
```

Componentes aos quais se aplica:

```
TThread
```

Dragging

Descrição

Esse método especifica se um componente está sendo arrastado ou não.

Declaração

```
function Dragging: Boolean;
```

Exemplo

O trecho de código a seguir verifica se um botão de rádio chamado RadioButton1 está ou não sendo arrastado, se o resultado do método Dragging for True ou False. Em caso positivo, lhe atribui a cor amarela.

```
if RadioButton1.Dragging then RadioButton1.Color := clYellow;
```

Componentes aos quais se aplica:

Todos os controles.

DragLock

Descrição

Esse método associa a imagem que está sendo arrastada com o handle passado no parâmetro Window e a desenha na posição definida pelos parâmetros XPos e YPos.

Declaração

```
function DragLock(Window: HWND; XPos, YPos: Integer): Boolean;
```

Componentes aos quais se aplica:

```
TListImage
```

DragUnLock

Descrição

Esse método remove uma associação feita pelo método DragLock.

Declaração

```
procedure DragUnLock;
```

Componentes aos quais se aplica:

```
TListImage
```

Draw

Descrição

Esse método desenha um objeto gráfico (definido no parâmetro Graphic) nas coordenadas (X,Y) do canvas corrente.

Declaração

```
procedure Draw(X, Y: Integer; Graphic: TGraphic);
```

Componentes aos quais se aplica:

```
TCanvas
```

DrawFocusRect

Descrição

Esse método desenha um retângulo de foco definido pelo parâmetro Rect.

Declaração

```
procedure DrawFocusRect(const Rect: TRect);
```

Componentes aos quais se aplica:

```
TCanvas
```

DrawOverlay

Descrição

Esse método desenha uma imagem e um overlay no objeto do tipo TCanvas passado pelo parâmetro Canvas.

Declaração

```
procedure DrawOverlay(Canvas: TCanvas; X, Y: Integer; ImageIndex: Integer; Overlay: TOverlay);
```

Componentes aos quais se aplica:

```
TListImage
```

DropConnections

Descrição

O método DropConnections desfaz todas as conexões com bancos de dados inativos.

Declaração

```
procedure DropConnections;
```

Exemplo

O trecho de código a seguir desfaz todas as conexões com bancos de dados inativos.

```
Session. DropConnections;;
```

Componentes aos quais se aplica:

```
TSession
```

DropDatabase

Descrição

Esse método remove o banco de dados representado pelo componente.

Declaração

```
procedure DropDatabase;
```

Componentes aos quais se aplica:

```
TIBDatabase
```

DropDown

Descrição

Esse método faz com que seja exibida a lista drop-down correspondente ao controle, para que o usuário possa escolher um dos valores apresentados.

Declaração

```
procedure DropDown;
```

Exemplo

O trecho de código a seguir executa o método DropDown em um componente DBLookupCombo1 do tipo TDBLookupCombo.

```
DBLookupCombo1.DropDown;
```

Componentes aos quais se aplica:

```
TDBLookupCombo e TDBLookupComboBox
```

Edit

Descrição

Para componentes derivados da classe TDataset, esse método prepara o registro corrente do banco de dados para edição e atribui o valor dsEdit à propriedade State.

Para componentes do tipo TDataSource, se sua propriedade AutoEdit possuir o valor True e sua propriedade State possuir o valor dsBrowse, esse método executa o método de mesmo nome dos componentes derivados da classe TDataset.

Declaração

```
procedure Edit;
```

Componentes aos quais se aplica:

```
TDataSource, TClientDataset, TSimpleDataset, TIBClientDataset, TTable e TQuery
```

EditKey

Descrição

Esse método modifica o conteúdo do buffer de pesquisa.

Declaração
```
procedure EditKey;
```

Componentes aos quais se aplica:
TClientDataSet, TSimpleDataset, TIBClientDataset e TTable

EditRangeEnd

Descrição

Esse método permite que se modifique o valor definido pelo método SetRangeEnd.

Declaração
```
procedure EditRangeEnd;
```

Componentes aos quais se aplica:
TClientDataset, TSimpleDataset, TIBClientDataset e TTable

EditRangeStart

Descrição

Esse método permite que se modifique o valor definido pelo método SetRangeStart.

Declaração
```
procedure EditRangeStart;
```

Componentes aos quais se aplica:
TClientDataset, TSimpleDataset, TIBClientDataset e TTable

Eject

Descrição

O método Eject executa a ejeção em um dispositivo multimídia.

Declaração
```
procedure Eject;
```

Exemplo

O trecho de código a seguir faz com que um dispositivo multimídia acione seu método Eject com um botão chamado Eject do tipo TButton:

```
procedure TForm1.EjectClick(Sender: TObject);
begin
      MediaPlayer1.Eject;
end;
```

Componentes aos quais se aplica:
TMediaPlayer

Ellipse

Descrição

Esse método desenha no canvas uma elipse inscrita em um retângulo definido por (X1,Y1) – vértice superior esquerdo e (X2,Y2) – vértice inferior direito.

Declaração
```
procedure Ellipse(X1, Y1, X2, Y2: Integer);
```

Componentes aos quais se aplica:
TCanvas

EmptyTable

Descrição

Esse método deleta todos os registros da tabela especificada na propriedade TableName.

Declaração

```
procedure EmptyTable;
```

Componentes aos quais se aplica:

```
TClientDataset, TSimpleDataset, TIBClientDataset, TIBTable, TTable
```

EnableAlign

Descrição

Esse método habilita o realinhamento automático dos controles-filhos do controle corrente.

Declaração

```
procedure EnableAlign;
```

Componentes aos quais se aplica:

Todos os controles.

EnableControls

Descrição

Esse método restaura a conexão do banco de dados de todos os componentes a ele associados (do tipo TDataSource) que haviam sido desconectados por uma chamada ao método DisableControls.

Declaração

```
procedure EnableControls;
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery,TStoredProc, TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc,
TClientDataset, TSimpleDataset e TIBClientDataset
```

EndDoc

Descrição

Para componentes do tipo TQRPrinter, esse método indica o término do trabalho de impressão de um relatório e atribui o valor mbFinished à propriedade Status do componente.

Para componentes do tipo TPrinter, indica o término do trabalho corrente de impressão.

Declaração

```
procedure EndDoc;
```

Componentes aos quais se aplica:

```
TQRPrinter e TPrinter
```

EndDrag

Descrição

O método EndDrag encerra o processo de arrastar o controle. Se o valor da variável Drop é True, o controle é solto sobre um controle receptor.

Declaração

```
procedure EndDrag(Drop: Boolean);
```

Exemplo

O trecho de código a seguir encerra o processo de arrastar um botão chamado Button1 e o solta sobre o seu componente.

```
Button1.EndDrag(True);
```

Componentes aos quais se aplica:

Todos os controles.

EndUpdate

Descrição

Esse método atualiza a exibição de um objeto dos tipos TStrings, TStringList ou TOutline.

Declaração

```
procedure EndUpdate;
```

Exemplo

O trecho de código a seguir faz com que um componente chamado Outline1 execute o seu método EndUpdate:

```
Outline1.EndUpdate;
```

Componentes aos quais se aplica:

```
TStringList, TStrings e TOutline
```

EraseSection

Descrição

Esse método apaga uma seção (definida no parâmetro Section) no arquivo INI.

Declaração

```
procedure EraseSection(const Section: string);
```

Componentes aos quais se aplica:

```
TIniFile
```

Exchange

Descrição

Esse método muda a posição de dois itens de uma lista de objetos, definidos por seus índices nos parâmetros Index1 e Index2.

Declaração

```
procedure Exchange(Index1, Index2: Integer);
```

Componentes aos quais se aplica:

```
TList, TStrings e TOutline
```

ExecProc

Descrição

Esse método executa o procedimento armazenado no servidor.

Declaração

```
procedure ExecProc;
```

Componentes aos quais se aplica:

```
TADOStoredProc, TIBStoredProc, TSQLStoredProc  e TStoredProc
```

ExecQuery

Descrição

Esse método executa a query definida pelo componente.

Declaração

```
procedure ExecQuery;
```

Componentes aos quais se aplica:

```
TIBSQL
```

ExecSQL

Descrição

Esse método executa uma declaração SQL definida na propriedade SQL do componente. Use-o para declarações dos tipos INSERT, UPDATE, DELETE.

Declaração

```
procedure ExecSQL;
```

Componentes aos quais se aplica:

```
TIBSQL, TIBQuery, TADOQuery, TSQLQuery e TQuery
```

Execute

Descrição

Para componentes do tipo TBatchMove, esse método realiza a operação de transferência de dados da tabela-origem (especificada na sua propriedade Source) para a tabela-destino (especificada na sua propriedade Destination).

Para componentes dos tipos TColorDialog, TFontDialog, TOpenDialog, TOpenPictureDialog, TPrintDialog, TPrintSetupDialog e TSaveDialog, TSavePictureDialog exibe a caixa de diálogo e define se o usuário pressionou o botão OK.

Para objetos da classe TAction, executa o procedimento associado ao evento OnExecute.

Para objetos da classe TThread, define o código a ser executado quando a thread é inicializada.

Para objetos da classe TADOCommand, executa o comando definido na sua propriedade CommandText.

Para objetos da classe TSQLConnection, executa um comando SQL no banco de dados.

Para objetos da classe TRvProject, inicia a impressão do projeto de relatório representado pelo componente.

Declaração

Para objetos da classe TAction:

```
function Execute: Boolean; override;
```

Para componentes do tipo TADOCommand, este método é sobrecarregado com as seguintes declarações:

```
function Execute: _RecordSet; overload;
function Execute(const Parameters: OleVariant): _Recordset; overload;
function Execute(var RecordsAffected: Integer; var Parameters: OleVariant; ExecuteOptions:
TExecuteOption[]): _RecordSet; overload;
```

Para componentes dos tipos TbatchMove, TPrinterSetupDialog e TRvProject:

```
procedure Execute;
```

Para componentes do tipo TColorDialog, TFontDialog, TOpenDialog, TOpenPictureDialog, TPrintDialog, TSaveDialog e TSavePictureDialog:

```
function Execute: Boolean;
```

Para objetos da classe TThread:

```
procedure Execute; virtual; abstract;
```

Para objetos da classe TSQLConnection:

```
function Execute(const SQL: string; Params: TParams
; ResultSet:Pointer=nil): Integer;
```

Exemplo

O trecho de código a seguir faz com que um componente TBatchMove execute o seu método Execute:

```
BatchMove1.Execute;
```

Componentes aos quais se aplica:

```
TADOCommand, TAction, TBatchMove, TThread, TPrinterSetupDialog, TColorDialog, TFontDialog,
TOpenDialog, TOpenPictureDialog, TPrintDialog e TSaveDialog, TsavePictureDialog e
TSQLConnection
```

ExecuteAction

Descrição

Esse método executa o procedimento associado ao evento OnExecute do componente.

Declaração

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

Componentes aos quais se aplica:

```
TAction'List
```

ExecuteMacro

Descrição

Esse método tenta executar uma macro, definida em uma string, em uma aplicação servidora. O parâmetro WaitFlg define se sua aplicação deve aguardar a execução da macro na aplicação servidora.

Declaração

```
function ExecuteMacro(Cmd: PChar; WaitFlg: Boolean): Boolean;
```

Componentes aos quais se aplica:

```
TDDEClientConv
```

ExecuteMacroLines

Descrição

Esse método tenta executar uma macro, definida em uma lista de strings, em uma aplicação servidora. O parâmetro WaitFlg define se sua aplicação deve aguardar a execução da macro na aplicação servidora.

Declaração

```
function ExecuteMacroLines(Cmd: TStrings; WaitFlg: Boolean): Boolean;
```

Componentes aos quais se aplica:

```
TDDEClientConv
```

ExecuteReport

Descrição

Esse método inicia a impressão do projeto de relatório representado pelo componente, e cujo nome é passado como parâmetro.

Declaração

```
procedure ExecuteReport(ReportName: string);
```

Componentes aos quais se aplica:

```
TRvProject
```

Expand

Descrição

Para objetos do tipo TOutlineNode, esse método abre um item do tipo TOutlineNode, atribuindo o valor True à sua propriedade Expanded.

Para objetos do tipo TList, esse método aumenta a capacidade de armazenamento de uma lista de objetos, alterando o valor da sua propriedade Capacity.

Declaração

Para objetos do tipo TOutlineNode:

procedure Expand;

Para objetos do tipo TList:

function Expand: TList;

Componentes aos quais se aplica:

```
TList e TOutlineNode
```

FieldAddress

Descrição

Esse método retorna em um ponteiro o endereço do objeto definido pelo parâmetro Name.

Declaração

```
function FieldAddress(const Name: ShortString): Pointer;
```

Componentes aos quais se aplica:

Todos os objetos.

FieldByName

Descrição

Esse método retorna o objeto do tipo TField especificado no parâmetro FieldName. Se não for encontrado, gera um erro de exceção.

Declaração

```
function FieldByName(const FieldName: string): TField;
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery, TIBSQL,
TIBStoredProc, TTable, TQuery, TStoredProc, TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc,
TClientDataset, TSimpleDataset e TIBClientDataset
```

FileLoad

Descrição

Esse método carrega um arquivo do tipo TResType em um objeto do tipo TImageList. O parâmetro MaskColor define a cor da máscara transparente da imagem.

Declaração

```
function FileLoad(ResType: TResType; Name: string; MaskColor: TColor): Boolean;
```

Componentes aos quais se aplica:

```
TImageList
```

FillRect

Descrição

Esse método preenche a área retangular do canvas definida pelo parâmetro Rect com o pincel corrente.

Declaração

```
procedure FillRect(const Rect: TRect);
```

Componentes aos quais se aplica:

```
TCanvas
```

FindComponent

Descrição

O método FindComponent retorna o componente pertencente à array Components cuja propriedade Name é igual à definida no parâmetro AName.

Declaração

```
function FindComponent(const AName: string): TComponent;
```

Exemplo

Coloque vários componentes em um formulárioForm1, incluindo um componente Edit1 do tipo TEdit e um botão Button1. Quando o usuário selecionar o botão Button1, o valor da propriedade ComponentIndex de Edit1 será exibido pelo próprio componente Edit1:

```
procedure TForm1.Button1Click(Sender: TObject);
var
      TheComponent: TComponent;
begin
      TheComponent := FindComponent('Edit1');
      Edit1.Text := IntToStr(TheComponent.ComponentIndex);
end;
```

Componentes aos quais se aplica:

Todos os componentes.

FindDatabase

Descrição

Para objetos da classe TIBTransaction, retorna o índice, na sua propriedade Databases, do objeto da classe TIBDatabase, passado como parâmetro.

Para objetos da classe TSession, retorna o índice, na sua propriedade Databases, do objeto da classe TDatabase, cujo nome é passado como parâmetro.

Declaração

Para objetos da classe TIBTransaction:

```
function FindDatabase (db: TIBDatabase): Integer;
```

Para objetos da classe TSession:

```
function FindDatabase(const DatabaseName: String): TDatabase;
```

Componentes aos quais se aplica:

TSession, TIBTransaction

FindField

Descrição

Esse método retorna o objeto do tipo TField especificado no parâmetro FieldName. Se não for encontrado, retorna nil.

Declaração

```
function FindField(const FieldName: string): TField;
```

Componentes aos quais se aplica:

TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery, TIBStoredProc, TTable, TQuery, TStoredProc, TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc, TClientDataset, TSimpleDataset e TIBClientDataset

FindKey

Descrição

Esse método pesquisa todos os registros em uma tabela até encontrar aquele cujos índices dos campos correspondam aos definidos no parâmetro KeyValues.

Declaração

```
function FindKey(const KeyValues: array of const): Boolean;
```

Componentes aos quais se aplica:

TClientDataSet, TIBClientDataset, TSimpleDataset e TTable

FindNearest

Descrição

Esse método pesquisa todos os registros em uma tabela até encontrar o próximo registro cujos índices dos campos forem iguais ou superiores aos definidos no parâmetro KeyValues.

Declaração

```
procedure FindNearest(const KeyValues: array of const);
```

Componentes aos quais se aplica:

TClientDataSet, TIBClientDataset, TSimpleDataset e TTable

FindNextPage

Descrição

Esse método retorna a próxima página do controle. Se o parâmetro GoForWard for igual a True, a próxima página será a página numericamente subseqüente. Se o parâmetro GoForWard for igual a False, a próxima página será a página numericamente anterior.

Declaração

```
function FindNextPage(CurPage: TTabSheet; GoForward, CheckTabVisible: Boolean): TTabSheet
```

Componentes aos quais se aplica:
```
TPageControl
```

FindTransaction

Descrição

Esse método retorna o índice da transação representada por um objeto da classe TIBTransaction passado como parâmetro.

Declaração
```
function FindTransaction (TR: TIBTransaction): Integer;
```

Componentes aos quais se aplica:
```
TIBDatabase
```

First

Descrição

Para componentes do tipo TTable, TStoredProc e TQuery, esse método move o ponteiro

para o primeiro registro do banco de dados.

Para componentes do tipo TList, esse método retorna um ponteiro que aponta para o primeiro objeto da lista.

Declaração

Para componentes dos tipos derivados da classe TDataset:
```
procedure First;
```

Para componentes do tipo TList:
```
function First: Pointer;
```

Componentes aos quais se aplica:
```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TClientDataset, TIBClientDataset,
TIBDataset, TIBTable, TIBQuery, TIBStoredProc, TTable, TQuery, TStoredProc, TSimpleDataset,
TSQLDataset, TSQLTable, TSQLQuery e TSQLStoredProc
```

FloodFill

Descrição

Esse método preenche a superfície da tela com o pincel corrente, começando no ponto (X,Y) e espalhando-se por todas as direções até encontrar a cor definida com o contorno (parâmetro Color).

Declaração
```
procedure FloodFill(X, Y: Integer; Color: TColor; FillStyle: TFillStyle);
```

Componentes aos quais se aplica:
```
TCanvas
```

FocusControl

Descrição

Esse método coloca o foco da aplicação no primeiro controle associado ao banco de dados que contém o campo que chama o método.

Declaração
```
function FocusControl;
```

Exemplo

O trecho de código a seguir faz com que um campo StringField1, do tipo TStringField, acione seu método FocusControl:

```
StringField1.FocusControl;
```

Componentes aos quais se aplica:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

Focused

Descrição

O método Focused informa se o controle possui o foco da aplicação, isto é, se ele é o controle ativo.

Declaração

```
function Focused: Boolean;
```

Exemplo

O trecho de código a seguir verifica se um controle chamado Edit1 do tipo TEdit possui o foco da aplicação quando o usuário tenta encerrá-la. Se Edit1 possuir o foco, será exibida uma mensagem.

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
        If Edit1.Focused then ShowMessage('Edit1 possui o foco');
end;
```

Componentes aos quais se aplica:

Todos os controles.

FrameRect

Descrição

Esse método desenha um retângulo definido pelo parâmetro Rect, pintando a sua borda com o pincel corrente.

Declaração

```
procedure FrameRect(const Rect: TRect);
```

Componentes aos quais se aplica:

```
TCanvas
```

Free

Descrição

Esse método destrói um componente, liberando toda a memória a ele alocada. Este apresenta uma vantagem sobre o método Destroy, pois não gera um erro quando o objeto não existe.

Declaração

```
procedure Free;
```

Exemplo

A linha de código a seguir mostra um botão Button1 do tipo TButton executando o seu método Free.

```
Button1.Free;
```

Componentes aos quais se aplica:

Todos os objetos, controles e componentes.

FreeBookmark

Descrição

Esse método libera os recursos do sistema reservados por uma chamada ao método GetBookmark.

Declaração

```
procedure FreeBookmark(Bookmark: TBookmark);
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc,TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc,
TClientDataset, TSimpleDataset e TIBClientDataset
```

FreeInstance

Descrição

Esse método libera a memória alocada pelo método NewInstance, após chamar o método IntanceSize.

Declaração

```
procedure FreeInstance; virtual;
```

Componentes aos quais se aplica:

Todos os objetos.

FreeNotification

Descrição

Esse método notifica o componente definido no parâmetro AComponent antes de destruí-lo.

Declaração

```
procedure FreeNotification(AComponent: TComponent);
```

Componentes aos quais se aplica:

Todos os componentes.

FullCollapse

Descrição

Esse método oculta todos os subitens em um componente do tipo TOutline.

Declaração

```
procedure FullCollapse;
```

Exemplo

O trecho de código a seguir faz com que um componente Outline1 execute o seu método FullCollapse:

```
Outline1. FullCollapse;
```

Componentes aos quais se aplica:

```
TOutline
```

FullExpand

Descrição

Esse método exibe todos os subitens em um componente do tipo TOutlineNode ou todos os subitens de todos os itens de um componente TOutline.

Descrição

```
procedure FullExpand;
```

Exemplo

O trecho de código a seguir faz com que um componente Outline1 execute o seu método FullExpand:

```
Outline1. FullExpand;
```

Componentes aos quais se aplica:

```
TOutline
```

GetAliasNames

Descrição

Esse método armazena, em uma lista de strings, os nomes dos aliases definidos no Borland DataBase Engine.

Declaração

```
procedure GetAliasNames(List: TStrings);
```

Exemplo

O trecho de código a seguir armazena, em uma lista de strings chamada Lista_Aliases, os nomes dos aliases definidos no Borland DataBase Engine:

```
Session.GetAliasNames(Lista_aliases);
```

Componentes aos quais se aplica:

```
TSession
```

GetAliasParams

Descrição

Esse método armazena, em uma lista de strings, os parâmetros associados a um alias do BDE definido pelo parâmetro AliasName.

Declaração

```
procedure GetAliasParams(const AliasName: string; List: TStrings);
```

Componentes aos quais se aplica:

```
TSession
```

GetAsHandle

Descrição

Esse método retorna um handle para o objeto armazenado no clipboard.

Declaração

```
function GetAsHandle (Format: Word): THandle;
```

Componentes aos quais se aplica:

```
TClipBoard
```

GetBitmap

Descrição

Esse método retorna a imagem especificada no parâmetro Index como um bitmap no parâmetro Image do tipo TBitmap.

Declaração

```
procedure GetBitmap(Index: Integer; Image: TBitmap);
```

Componentes aos quais se aplica:

```
TImageList
```

GetBookmark

Descrição

Esse método salva a informação de um registro de um banco de dados, de forma a poder acessá-lo por uma chamada posterior ao método GotoBookmark.

Declaração

```
procedure GetBookmark(Bookmark: TBookmark);
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc,
TClientDataset, TSimpleDataset e TIBClientDataset
```

GetChildren

Descrição

Esse método executa o procedimento passado como parâmetro para cada uma das ações definidas no componente (definidas como objetos da classe TAction).

Declaração

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

Componentes aos quais se aplica:

```
TActionList
```

GetComponent

Descrição

Esse método retorna um componente armazenado no clipboard e o ajusta de acordo com os parâmetros Owner e Parent.

Declaração

```
function GetComponent(Owner, Parent: TComponent): TComponent;
```

Componentes aos quais se aplica:

```
TClipBoard
```

GetData

Descrição

Esse método retorna no parâmetro Buffer o dado armazenado no campo. Se o buffer não tiver espaço suficiente para armazenar o dado, retorna False.

Declaração

```
function GetData(Buffer: Pointer): Boolean;
```

Exemplo

O trecho de código a seguir transfere o dado de um campo StringField1, do tipo TStringField, para um buffer temporário:

```
with StringField1 do
begin
     GetMem(Buffer, DataSize);
      if not Field1.GetData(Buffer) then ShowMessage(FieldName + ' is NULL');
end;
```

Componentes aos quais se aplica:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

GetDatabaseNames

Descrição

Esse método armazena, em uma lista de strings, os nomes dos bancos de dados e respectivos aliases definidos no Borland DataBase Engine para a aplicação.

Declaração

```
procedure GetDatabaseNames(List: TStrings);
```

Exemplo

O trecho de código a seguir armazena, em uma lista de strings denominada Lista, os nomes dos bancos de dados e respectivos aliases definidos no Borland DataBase Engine para a aplicação:

```
Session.GetDatabaseNames(Lista);
```

Componentes aos quais se aplica:

```
TSession
```

GetDataItem

Descrição

O método GetDataItem retorna o índice do primeiro item de um componente do tipo TOutline em um dado especificado no parâmetro Value.

Declaração

```
function GetDataItem(Value: Pointer): Longint;
```

Componentes aos quais se aplica:

```
TOutline
```

GetDeltaPacket

Descrição

Esse método retorna um pacote delta de informações a partir de uma solicitação http..

Declaração

```
function GetDelta(Request: TWebRequest
): string;
```

Componentes aos quais se aplica:

```
TXMLBroker
```

GetDetailLinkFields

Descrição

Esse método armazena, em dois objetos da classe TList, referências aos objetos que representam os campos utilizados em um relacionamento.

Declaração

```
GetDetailLinkFields(MasterFields, DetailFields: TList); override;
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc,
TClientDataset, TSimpleDataset e TIBClientDataset
```

GetDetailSQL

Descrição

Esse método retorna uma Declaração SQL capaz de produzir o mesmo resultado exibido pelo componente.

Declaração
```
function GetDetailSQL(ValueArray: TSmallIntArray; SelectList: string; bActive: Boolean): string;
```

Componentes aos quais se aplica:
```
TDecisionCube
```

GetDriverNames

Descrição

Esse método armazena, em uma lista de strings, os nomes dos drivers atualmente instalados no Borland Database Engine, exceto os do 'PARADOX' e 'DBASE', que são manipulados pelo driver 'STANDARD'.

Declaração
```
procedure GetDriverNames(List: TStrings);
```

Exemplo

O trecho de código a seguir armazena, em uma lista de strings Lista, os nomes dos drivers atualmente instalados no Borland Database Engine:
```
Session.GetDriverNames(Lista);
```

Componentes aos quais se aplica:
```
TSession
```

GetDriverParams

Descrição

Esse método armazena, em uma lista de strings, os nomes dos parâmetros usados pelo driver do Borland Database Engine especificado no parâmetro DriverName.

Declaração
```
procedure GetDriverParams(const DriverName: string;
List: TStrings);
```

Componentes aos quais se aplica:
```
TSession
```

GetErrorCount

Descrição

Esse método retorna o número de erros no pacote delta recebido.

Declaração
```
function GetDelta(Request: TWebRequest): string;
```

Componentes aos quais se aplica:
```
TXMLBroker
```

GetFieldNames

Descrição

Esse método limpa a lista de strings definida no parâmetro List e adiciona a ela o nome de cada campo do banco de dados.

Declaração
```
procedure GetFieldNames(List: TStrings);
```

Componentes aos quais se aplica:
```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc,
TClientDataset, TSimpleDataset e TIBClientDataset
```

GetFirstChild

Descrição

Esse método retorna o índice do primeiro subitem de um item do tipo TOutlineNode.

Declaração

```
function GetFirstChild: Longint;
```

Componentes aos quais se aplica:

```
TOutlineNode
```

GetFormImage

Descrição

O método GetFormImage cria um bitmap com a imagem do formulário, como ele seria impresso.

Declaração

```
function GetFormImage: TBitmap;
```

Exemplo

O trecho de código a seguir copia, em um bitmap, a imagem de um formulário:

```
var
     Bitmap1 : TBitmap;
begin
     Bitmap1 := Form1.GetFormImage;
```

Componentes aos quais se aplica:

```
TForm
```

GetHotSpot

Descrição

Esse método retorna o hotspot associado à imagem sendo arrastada.

Declaração

```
function GetHotSpot: TPoint;
```

Componentes aos quais se aplica:

```
TImageList
```

GetIcon

Descrição

Esse método retorna a imagem especificada no parâmetro Index como um ícone no parâmetro Image do tipo TIcone.

Declaração

```
procedure GetIcon(Index: Integer; Image: TIcon);
```

Componentes aos quais se aplica:

```
TImageList
```

GetImageBitmap

Descrição

Esse método retorna um handle para um bitmap que contém todas as imagens do componente.

Declaração

```
function GetImageBitmap: HBITMAP;
```

Componentes aos quais se aplica:
```
TImageList
```

GetIndexForPage

Descrição

O método GetIndexForPage retorna o valor da propriedade PageIndex da página cuja string de exibição, definida na propriedade Strings, é especificada no parâmetro PageName.

Declaração
```
function GetIndexForPage(const PageName: string): Integer;
```

Exemplo

O trecho de código a seguir faz com que um componente chamado Label1 do tipo TLabel exiba o índice da página cujo nome, definido na propriedade Strings, é igual a 'Exemplo'.
```
Label1.Caption := IntToStr(TabbedNotebook1.GetIndexForPage('Exemplo'));
```

Componentes aos quais se aplica:
```
TTabbedNotebook
```

GetIndexNames

Descrição

Esse método armazena, em uma lista de strings, os nomes dos índices definidos para a tabela.

Declaração
```
procedure GetIndexNames(List: TStrings);
```

Componentes aos quais se aplica:
```
TADOTable, TIBTable, TTable, TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc,
TClientDataset, TSimpleDataset e TIBClientDataset
```

GetItem

Descrição

O método GetItem retorna o índice do item de um componente do tipo TOutline que está situado nas coordenadas (X,Y) em pixels.

Declaração
```
function GetItem(X, Y: Integer): Longint;
```

Componentes aos quais se aplica:
```
TOutline
```

GetItemPath

Descrição

O método GetItemPath retorna em uma string o path de um diretório definido por um índice em um componente do tipo TDirectoryListBox. O primeiro diretório da lista possui índice igual a 0.

Declaração
```
function GetItemPath(Index : Integer): string;
```

Componentes aos quais se aplica:
```
TDirectoryListBox
```

GetLastChild

Descrição

Retorna o índice do último subitem de um item do tipo TOutlineNode.

Declaração
```
function GetLastChild: Longint;
```

Componentes aos quais se aplica:
```
TOutlineNode
```

GetMaskBitmap

Descrição

Esse método retorna um handle para um bitmap que contém todas as máscaras das imagens do componente.

Declaração
```
function GetMaskBitmap: HBITMAP;
```

Componentes aos quais se aplica:
```
TImageList
```

GetNamePath

Descrição

Esse método retorna o nome do objeto como aparece no Object Inspector.

Declaração
```
function  GetNamePath: string; dynamic;
```

Componentes aos quais se aplica:
```
TADOCommand
```

GetNextChild

Descrição

Esse método retorna o índice do próximo item do tipo TOutlineNode com o mesmo item-pai do item atual.

Declaração
```
function GetNextChild(Value: Longint): Longint;
```

Componentes aos quais se aplica:
```
TOutlineNode
```

GetParentComponent

Descrição

Esse método retorna o componente TActionList ao qual o objeto está vinculado.

Declaração
```
function GetParentComponent: TComponent; override;
```

Componentes aos quais se aplica:
```
TAction
```

GetPassword

Descrição

Esse método dispara um evento OnPassword (se existir) ou exibe a caixa de diálogo padrão de password. Retorna True, se o usuário selecionar o botão OK, e False, se o usuário selecionar o botão Cancel.

Declaração
```
function GetPassword: Boolean;
```

Exemplo

O trecho de código a seguir aciona o método GetPassword:

```
Session.GetPassword;
```

Componentes aos quais se aplica:

```
TSession
```

GetPrevChild

Descrição

Esse método retorna o índice do item anterior do tipo TOutlineNode com o mesmo item-pai do item atual.

Declaração

```
function GetPrevChild(Value: Longint): Longint;
```

Componentes aos quais se aplica:

```
TOutlineNode
```

GetPrinter

Descrição

Esse método retorna a impressora corrente.

Declaração

```
procedure GetPrinter (ADevice, ADriver, APort: PChar; var ADeviceMode: THandle);
```

Componentes aos quais se aplica:

```
TPrinter
```

GetProcedureNames

Descrição

Esse método armazena, em uma lista de strings, os nomes dos procedimentos armazenados definidos para o banco de dados acessado pelo componente.

Declaração

```
procedure GetProcedureNames(List: TStrings);
```

Componentes aos quais se aplica:

```
TADOConnection e TSQLConnection.
```

GetResults

Descrição

Esse método retorna os valores dos parâmetros de saída em um procedimento armazenado em um servidor do tipo Sybase.

Declaração

```
procedure GetResults;
```

Componentes aos quais se aplica:

```
TStoredProc
```

GetSelTextBuf

Descrição

O método GetSelTextBuf copia o texto selecionado de um controle no buffer apontado pela variável Buffer, até o número de caracteres definido no parâmetro BufSize. O texto colocado em buffer é uma string terminada em zero. O método retorna o número de caracteres copiados.

Declaração

```
function GetSelTextBuf(Buffer: PChar; BufSize: Integer): Integer;
```

Exemplo

O exemplo a seguir copia o texto selecionado em um componente TEdit chamado Edit1 em uma string terminada em zero, e depois coloca essa string em outro componente TLabel denominado Label1 quando o usuário dá um clique sobre um botão chamado Button1 do tipo TButton.

```
procedure TForm1.Button1Click(Sender: TObject);
var
 Buffer: PChar;
 Size: Integer;
begin
       Size := Edit1.SelLength;
       Inc(Size);
       GetMem(Buffer, Size);
       Edit1.GetSelTextBuf(Buffer,Size);
       Label1.Caption := StrPas(Buffer);
       FreeMem(Buffer, Size);
end;
```

Componentes aos quais se aplica:

```
TDBEdit, TDBMemo, TEdit, TMaskEdit e TMemo
```

GETSQL

Descrição

Esse método retorna uma Declaração SQL capaz de produzir o mesmo resultado exibido pelo componente.

Declaração

```
function GetSQL(ValueArray: TSmallIntArray; bActive: Boolean): string;
```

Componentes aos quais se aplica:

```
TDecisionCube
```

GETSTOREDPROCNAMES

Descrição

Esse método armazena, em uma lista de strings, os nomes das procedures definidas para um banco de dados SQL determinado no parâmetro DatabaseName (não se aplica ao dBASE e Paradox).

Declaração

```
procedure GeTStoredProcNames(const DatabaseName: string; List: TStrings);
```

Componentes aos quais se aplica:

```
TSession
```

GETTABLENAMES

Descrição

Esse método armazena, em uma lista de strings, os nomes das tabelas associadas ao banco de dados definido no parâmetro DatabaseName. O parâmetro Pattern limita os nomes de tabelas a serem procuradas.

Declaração

```
procedure GeTTableNames(const DatabaseName, Pattern: string;Extensions, SystemTables: Boolean;
List: TStrings);
```

Componentes aos quais se aplica:

```
TADOConnection, TSQLConnection, Tssession e TIBDatabase
```

GetTabOrderList

Descrição

Esse método retorna uma lista dos controles-filhos do controle corrente, ordenados pelo valor armazenado na sua propriedade TabOrder.

Declaração

```
procedure GetTabOrderList(List: TList);
```

Componentes aos quais se aplica:

```
Todos os controles.
```

GetText

Descrição

Esse método retorna uma lista de strings como uma string de terminação nula.

Declaração

```
function GetText: PChar;
```

Componentes aos quais se aplica:

```
TStrings e TStringList
```

GetTextBuf

Descrição

O método GetTextBuf copia o texto de um controle no buffer apontado pela variável Buffer, até o número de caracteres definido no parâmetro BufSize. O texto colocado em buffer é uma string terminada em zero. O método retorna o número de caracteres copiados.

Declaração

```
function GetTextBuf(Buffer: PChar; BufSize: Integer): Integer;
```

Exemplo

O exemplo a seguir copia o texto exibido em um componente TLabel denominado Label1 em uma string terminada em zero e depois coloca essa string em outro componente TLabel chamado Label2 quando o usuário dá um clique sobre um botão denominado Button1.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    Buffer: PChar;
    Size: Byte;
begin
    Size := Label1.GetTextLen;
    Inc(Size);
    GetMem(Buffer, Size);
    Label1.GetTextBuf(Buffer,Size);
    Label2.Caption := StrPas(Buffer);
    FreeMem(Buffer, Size);
end;
```

Componentes aos quais se aplica:

Todos os controles e objetos do tipo TClipboard.

GetTextLen

Descrição

O método GetTextLen retorna o comprimento do texto exibido por um controle.

Declaração

```
function GetTextLen: Integer;
```

Exemplo

A linha de código abaixo atribui a uma variável inteira Size o comprimento do texto exibido em um componente Edit1 do tipo TEdit.

```
Size := Edit1.GetTextLen;
```

Componentes aos quais se aplica:

Todos os controles.

GetXMLRecords

Descrição

Esse método retorna em uma string os registros do pacote delta enviado pelo servidor de aplicações.

Declaração

```
function GetXMLRecords(var RecsOut: Integer; var OwnerData: OleVariant; XMLOptions:
TXMLOptions): string;
```

Componentes aos quais se aplica:

```
TXMLBroker
```

GotoBookmark

Descrição

Esse método move o cursor para o registro definido em uma chamada a um método GetBookmark.

Declaração

```
procedure GotoBookmark(Bookmark: TBookmark);
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc,
TClientDataset, TSimpleDataset e TIBClientDataset
```

GotoCurrent

Descrição

Esse método sincroniza duas tabelas de um banco de dados.

Declaração

```
procedure GotoCurrent(Table: TTable);
```

Componentes aos quais se aplica:

```
TTable, TClientDataset, TSimpleDataset e TIBClientDataset
```

GotoKey

Descrição

Esse método é usado com os métodos SetKey e GetKey para se mover para um registro da tabela.

Declaração

```
function GotoKey: Boolean;
```

Componentes aos quais se aplica:

```
TTable, TClientDataset, TSimpleDataset e TIBClientDataset
```

GotoNearest

Descrição

Esse método é usado com os métodos SetKey e GetKey para se mover para o próximo registro cujos índices dos campos sejam iguais ou superiores aos definidos no parâmetro KeyValues.

Declaração

```
procedure GotoNearest;
```

Componentes aos quais se aplica:

```
TTable, TClientDataset, TSimpleDataset e TIBClientDataset
```

HandleAllocated

Descrição

Esse método retorna um valor booleano que indica se existe ou não um handle alocado para o controle.

Declaração

```
function HandleAllocated: Boolean;
```

Exemplo

O trecho de código a seguir verifica se existe um handle alocado para um componente GroupBox1 do tipo TGroupBox, exibindo-o como um componente Label1 do tipo TLabel. Caso o handle não tenha sido alocado, exibe uma mensagem.

```
var
      TheValue: string;
begin
      if GroupBox1.HandleAllocated then
            TheValue := IntToStr(GroupBox1.Handle)
      else TheValue := 'Handle não alocado.';
      Label1.Caption := TheValue;
end;
```

Componentes aos quais se aplica:

```
Todos os controles.
```

HandleException

Descrição

Esse método manipula as exceções de uma aplicação.

Declaração

```
procedure HandleException(Sender: TObject);
```

Exemplo

O trecho de código a seguir faz a manipulação default das exceções de uma aplicação.

```
try
     {Código gerador de exceções}
except
      Application.HandleException(Self);
end;
```

Componentes aos quais se aplica:

```
TApplication
```

HandleNeeded

Descrição

Esse método cria um handle para um controle, se ele já não possuir um.

Declaração

```
procedure HandleNeeded;
```

Exemplo

A linha de código a seguir cria um handle para um botão chamado Button1 do tipo

```
TButton:
Button1.HandleNeeded;
```

Componentes aos quais se aplica:

Todos os controles.

HasFormat

Descrição

Esse método determina se o objeto armazenado no clipboard possui o formato definido pelo parâmetro Format. Os valores possíveis para o parâmetro Format são CF_TEXT (texto), CF_BITMAP (gráfico de bitmap), CF_METAFILE (gráfico metafile), CF_PICTURE (objeto do tipo Tpicture) e CF_OBJECT (qualquer objeto persistente).

Declaração

```
procedure HasFormat(Format: Word): Boolean;
```

Componentes aos quais se aplica:

```
TClipboard
```

Hash

Descrição

Esse método retorna um valor que identifica a instância do objeto.

Declaração

```
function Hash(Maximum: Integer): Integer;
```

Interfaces às quais se aplica:

```
ICorbaObject
```

HelpCommand

Descrição

Esse método dá acesso aos comandos do arquivo WinHelp da API do Windows. O

comando é passado pelo parâmetro Command.

Declaração

```
function HelpCommand(Command: Word; Data: Longint): Boolean;
```

Exemplo

O trecho de código a seguir acessa o tópico Contents do Help On-line da API do Windows.

```
Application.HelpCommand(HELP_CONTENTS, 0);
```

Componentes aos quais se aplica:

```
TApplication
```

HelpContext

Descrição

Esse método chama o programa WinHelp, o sistema de Help On-line do Windows, desde que haja um arquivo de Help da aplicação definido na sua propriedade HelpFile. O parâmetro Context especifica o número da ID que define a tela de auxílio.

Declaração

```
function HelpContext(Context: THelpContext): Boolean;
```

Exemplo

O trecho de código a seguir faz com que a tela de auxílio, cuja ID é igual a 715 no arquivo definido na propriedade HelpFile, seja exibida.

```
Application.HelpContext(715);
```

Componentes aos quais se aplica:

```
TApplication
```

HelpJump

Descrição

Esse método chama o programa WinHelp, o sistema de Help On-line do Windows, desde que haja um arquivo de Help da aplicação definido na sua propriedade HelpFile. O parâmetro JumpID é a string de contexto que define a tela de auxílio a ser exibida.

Declaração

```
function HelpJump(const JumpID: string): Boolean;
```

Exemplo

O trecho de código a seguir faz com que a tela de auxílio, cuja string de contexto é igual à 'string de contexto' no arquivo definido na propriedade HelpFile, seja exibida.

```
Application.HelpJump('string de contexto');
```

Componentes aos quais se aplica:

```
TApplication
```

Hide

Descrição

O método Hide torna invisível um componente, fazendo sua propriedade Visible := False.

Declaração

```
procedure Hide;
```

Exemplo

Se você quiser que um botão chamado Button1 desapareça de um formulário que o contém (form1) ao se clicar sobre ele com o botão esquerdo do mouse, defina o seu evento OnClick da seguinte forma:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
       Button1.Hide;
end;
```

Componentes aos quais se aplica:

Todos os controles e componentes.

HideDragImage

Descrição

Esse método oculta a imagem correntemente arrastada.

Declaração

```
procedure HideDragImage;
```

Componentes aos quais se aplica:

```
TImageList
```

IndexOf

Descrição

Para componentes do tipo TmenuItem, esse método retorna a posição de um item de menu (especificado no parâmetro Item) em um menu. Para o primeiro item de um menu, IndexOf retorna 0. Se o item especificado não fizer parte do menu, retorna -1.

Para objetos dos tipos TStrings e TStringList, esse método retorna a posição da string definida pelo parâmetro S.

Declaração

Para componentes do tipo TMenuItem:

```
function IndexOf(Item: TMenuItem): Integer;
```

Para objetos dos tipos TStrings e TStringList:

```
function IndexOf(const S: string): Integer;
```

Exemplo

O trecho de código a seguir exibe, em um componente Label1 do tipo TLabel, o índice de um item de menu chamado Sair, de um menu Arquivo.

```
Label1.Caption := IntToStr(Arquivo.IndexOf(Sair));
```

Componentes aos quais se aplica:

```
TStrings, TStringList e TMenuItem
```

IndexOfObject

Descrição

Esse método retorna a posição do objeto definido pelo parâmetro AObject.

Declaração

```
function IndexOfObject(AObject: TObject): Integer;
```

Componentes aos quais se aplica:

```
TStrings e TStringList
```

InheritsFrom

Descrição

Esse método define se o objeto passado pelo parâmetro AClass é um ancestral do objeto corrente.

Declaração

```
class function InheritsFrom(AClass: TClass): Boolean;
```

Componentes aos quais se aplica:

Todos os objetos.

InitInstance

Descrição

Esse método cria uma nova instância de um objeto.

Declaração

```
class procedure InitInstance(Instance: Pointer): TObject;
```

Componentes aos quais se aplica:

Todos os objetos.

Insert

Descrição

Para componentes do tipo TOutline, esse método insere um item na posição especificada pelo parâmetro Index.

Para componentes do tipo TMenuItem, esse método insere um item de menu na posição especificada pelo parâmetro Index.

Para componentes derivados da classe TDataset, coloca o banco de dados no modo de inserção e insere um registro em branco na posição atual do cursor.

Para objetos do tipo Tlist, insere um item, definido pelo parâmetro Item, na posição especificada pelo parâmetro Index.

Para objetos dos tipos TStrings e TStringList, insere uma string, definida pelo parâmetro S,

na posição especificada pelo parâmetro Index.

Declaração

Para componentes do tipo TOutline:

```
function Insert(Index: Longint; const Text: string): Longint;
```

Para componentes do tipo TMenuItem:

```
procedure Insert(Index: Integer; Item: TMenuItem);
```

Para componentes derivados da classe TDataset:

```
procedure Insert;
```

Para objetos do tipo Tlist:

```
procedure Insert(Index: Integer; Item: Pointer);
```

Para objetos dos tipos TStrings e TStringList:

```
procedure Insert(Index: Integer; const S: string);
```

Exemplo

O trecho de código a seguir insere um item, na posição do item atualmente selecionado, em um componente Outline1 do tipo TOutline:

```
Outline1.Insert(Outline1.SelectedItem, 'Novo item');
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc,
TStrings, TStringList, Tlist e TMenuItem
```

InsertComponent

Descrição

Esse método insere um componente passado no parâmetro AComponent na lista de componentes definida pela propriedade Components do componente atual.

Declaração

```
procedure InsertComponent(AComponent: TComponent);
```

Exemplo

O trecho de código a seguir insere um botão Button1 do tipo TButton no array de componentes de um formulário chamado Form1.

```
Form1.InsertComponent(Button1);
```

Componentes aos quais se aplica:

Todos os componentes.

InsertControl

Descrição

Esse método insere um controle passado no parâmetro AControl na lista de controles definida pela propriedade Controls do controle atual.

Declaração

```
procedure InsertControl(AControl: TControl);
```

Exemplo

O trecho de código a seguir insere um botão Button1 do tipo TButton na array de controles de um formulário chamado Form1.

```
Form1.InsertControl(Button1);
```

Componentes aos quais se aplica:

Todos os controles.

InsertIcon

Descrição

Esse método insere um ícone no componente após a imagem definida pelo parâmetro Index.

Declaração

```
procedure InsertIcon(Index: Integer; Image: TIcon);
```

Componentes aos quais se aplica:

```
TImageList
```

InsertMasked

Descrição

Esse método insere um bitmap no componente após a imagem definida pelo parâmetro Index, criando uma máscara definida pelo parâmetro MaskColor.

Declaração

```
procedure InsertMasked(Index: Integer; Image: TBitmap; MaskColor: TColor);
```

Componentes aos quais se aplica:

```
TImageList
```

InsertObject

Descrição

Esse método insere um item contendo dados na posição especificada pelo parâmetro Index em um componente do tipo TOutline.

Declaração

```
function InsertObject(Index: Longint; const Text: string; const Data: Pointer): Longint;
```

Exemplo

O trecho de código a seguir insere um objeto chamado Bitmap1 do tipo TBitmap na posição do item atualmente selecionado em um componente Outline1 do tipo TOutline:

```
Outline1.InsertObject(Outline1.SelectedItem, 'Novo item', Bitmap1);
```

Componentes aos quais se aplica:

```
TOutline
```

InsertRecord

Descrição

Esse método insere um novo registro no banco de dados utilizando os valores de campos passados pelo parâmetro Values.

Declaração

```
procedure InsertRecord(const Values: array of const);
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TSQLDataset, TSQLTable, TSQLQuery, TSQLStoredProc,
TStrings, TStringList, Tlist e TMenuItem
```

InstanceSize

Descrição

Esse método retorna o tamanho, em bytes, de cada instância de um objeto.

Declaração

```
class function InstanceSize: Longint;
```

Componentes aos quais se aplica:

Todos os objetos.

InValidate

Descrição

Esse método faz com que o conteúdo de um controle seja redesenhado.

Declaração

```
procedure Invalidate;
```

Componentes aos quais se aplica:

Todos os controles e componentes do tipo TForm.

IsA

Descrição

Esse método verifica se o objeto associado é do tipo passado como parâmetro.

Declaração

```
function IsA(const LogicalTypeId: string): Boolean;
```

Interfaces às quais se aplica:

```
ICorbaObject
```

ISEMPTY

Descrição

Esse método determina se a tabela representada pelo componente está vazia.

Declaração

```
function IsEmpty: Boolean;
```

Componentes aos quais se aplica:

```
TADOTable, TIBTable, TSQLTable, TTable, TADOQuery, TIBQuery
```

ISLOCAL

Descrição

Esse método retorna um inteiro diferente de zero se o objeto CORBA referenciado pela

interface for instanciado localmente.

Declaração

```
function IsLocal: CorbaBoolean; stdcall;
```

Interfaces às quais se aplica:

```
ICorbaObj
```

ISVALIDCHAR

Descrição

Esse método determina se o caractere digitado em um controle que representa o campo é caractere válido.

Declaração

```
function IsValidChar(InputChar: Char): Boolean; virtual;
```

Componentes aos quais se aplica:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

ITEMATPOS

Descrição

Para componentes dos tipos TDBListBox, TDirectoryListBox, TFileListBox e TListBox, esse método retorna o índice do item de uma caixa de listagem situado na posição especificada por um parâmetro Pos, do tipo Tpoint, ou -1, se não existir um item naquele ponto.

Para componentes do tipo TTabSet, esse método retorna o índice da guia situada na posição definida por dois parâmetros inteiros X e Y.

Declaração

Para componentes dos tipos TDBListBox, TDirectoryListBox, TFileListBox eTListBox:

```
function ItemAtPos(Pos: TPoint; Existing: Boolean): Integer;
```

Para componentes do tipo TTabSet:

```
function ItemAtPos(Pos: TPoint; Existing: Boolean): Integer;
```

Exemplo

O trecho de código a seguir seleciona a guia que está nas coordenadas (100,1) de

```
TabSet1:
TbSet1.TabIndex := TabSet1.ItemAtPos(100, 1);
```

Componentes aos quais se aplica:

```
TDBListBox, TDirectoryListBox, TFileListBox, TListBox e TabSet
```

ItemRect

Descrição

Esse método retorna o retângulo circunscrito a um item, especificado no parâmetro Item.

Declaração

```
function ItemRect(Item: Integer): TRect;
```

Exemplo

O trecho de código a seguir copia o retângulo circunscrito ao primeiro item de uma variável chamada ListBox1, do tipo TListBox, em uma variável CircRect do tipo Trect:

```
CircRect := ListBox1.ItemRect(0);
```

Componentes aos quais se aplica:

```
TDBListBox, TDirectoryListBox, TDrawGrid, TFileListBox, TListBox, TStringGrid e TTabSet
```

Last

Descrição

Para componentes derivados da classe TDataset, esse método move o ponteiro para o último registro do banco de dados.

Para objetos do tipo TList, esse método retorna o ponteiro para o último objeto da lista.

Declaração

Para componentes derivados da classe TDataset:

```
procedure Last;
```

Para componentes do tipo TList:

```
function Last: Pointer;
```

Componentes aos quais se aplica:

```
TList, TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TSimpleDataset, TSQLDataset, TSQLTable, TSQLQuery
e TSQLStoredProc
```

LineTo

Descrição

Esse método desenha uma linha da posição corrente até o ponto (X,Y).

Declaração

```
procedure LineTo(X, Y: Integer);
```

Componentes aos quais se aplica:

```
TCanvas
```

Load

Descrição

Esse método recupera a impressão de um relatório que foi direcionada para um arquivo.

Declaração

```
procedure Load(Filename: String)
```

Componentes aos quais se aplica:

```
TQRPrinter
```

LoadFromFile

Descrição

Esse método lê o arquivo especificado no parâmetro FileName e carrega seus dados no objeto ou componente.

No caso de componentes do tipo TRvProject, carrega o relatório a partir do arquivo especificado como parâmetro.

Declaração

```
procedure LoadFromFile(const FileName: string);
```

Exemplo

O trecho de código a seguir carrega o arquivo autoexec.bat em um componente TStrings que, nesse caso, é a propriedade Lines de um componente chamado Memo1 do tipo

```
TMemo.

Memo1.Lines.LoadFromFile('c:\autoexec.bat.');

NOTA = Note que, nesse caso, LoadFromFile é um método de Lines, que é um componente do tipo
TStrings. Por outro lado, Lines é ainda uma propriedade de TMemo.
```

Componentes aos quais se aplica:

```
TClientDataset, TDBMemoBuf, TIBClientDataset, TSimpleDataset, TBitmap, TBlobField,
TGraphicField, TMemoBuf, TMemoField, TGraphic, TIcon, TMetaFile, TPicture, TOutline,
TrvProject, TStringList e TStrings
```

LoadFromStream

Descrição

Esse método lê a stream especificada no parâmetro Stream e carrega seu conteúdo no

```
Campo ou no relatório.
```

Declaração

```
procedure LoadFromStream(Stream: TStream);
```

Exemplo

O trecho de código a seguir carrega uma stream chamada Stream1 em um campo BlobField1 do tipo TBlobField.

```
BlobField1.LoadFromStream(Stream1);
```

Componentes aos quais se aplica:

```
TBlobField, TClientDataset, TDBMemoBuf, TIBClientDataset, TGraphicField, TMemoBuf, TMemoField,
TRvProject e TSimpleDataset
```

LoadMemo

Descrição

Esse método copia um texto em um controle do tipo TDBMemo.

Declaração

```
procedure LoadMemo;
```

Exemplo

O trecho de código a seguir copia o texto armazenado em um campo do tipo TBLOBField em um componente DBMemo1 do tipo TDBMemo.

```
DBMemo1.LoadMemo;
```

Componentes aos quais se aplica:

```
TDBMemo
```

LoadPicture

Descrição

Esse método copia a imagem definida pela propriedade Picture para o controle.

Declaração

```
procedure LoadPicture;
```

Exemplo

O trecho de código a seguir copia a imagem definida na propriedade Picture para ser exibida pelo controle.

```
DBImage1.LoadPicture;
```

Componentes aos quais se aplica:

```
TDBImage
```

Locate

Descrição

Esse método permite uma pesquisa por campos não-indexados de uma tabela.

Declaração

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean;
```

Componentes aos quais se aplica:

```
TClientDataset, TIBClientDataset, TSimpleDataset, TADOTable, TADOStoredproc, TIBTable,
TIBStoredproc, TTable, TStoredProc
```

LockTable

Descrição

Esse método impede que outras aplicações acessem a tabela para leitura e/ou gravação.

Declaração

```
procedure LockTable(LockType: TLockType);
```

onde TLockType é um tipo enumerado definido da seguinte maneira:

```
type TLockType = (ltReadLock, ltWriteLock);
```

Componentes aos quais se aplica:

```
TTable
```

Merge

Descrição

Esse método mescla os menus de diversos formulários em uma aplicação não-MDI. O parâmetro Menu define o menu a ser mesclado.

Declaração

```
procedure Merge(Menu: TMainMenu);
```

Componentes aos quais se aplica:
`TMainMenu`

MessageBox

Descrição

Esse método encapsula a função MessageBox da API do Windows, sem que seja necessário especificar um handle. Esse método exibe uma mensagem e um conjunto de botões. O texto da mensagem, passado no parâmetro Text, não pode ter mais do que 255 caracteres. O parâmetro Caption define o texto a ser exibido na barra de títulos do quadro de mensagem (pode ter mais do que 255 caracteres). O parâmetro Flags define os botões que devem ser exibidos (veja maiores informações no arquivo de auxílio da API do Windows – WINAPI.HLP).

Declaração
```
function MessageBox(Text, Caption: PChar; Flags: Word): Integer;
```

Exemplo

O trecho de código a seguir exibe uma mensagem com um botão OK e um botão Cancel:
```
Application.MessageBox('Bem-vindo ao Delphi!', 'Mensagem', mb_OKCancel);
```

Componentes aos quais se aplica:
`TApplication`

MethodAddress

Descrição

Esse método retorna o endereço do método do tipo published especificado no parâmetro Name.

Declaração
```
class function MethodAddress(const Name: ShortString): Pointer;
```

Componentes aos quais se aplica:

Todos os objetos.

MethodName

Descrição

Esse método retorna o nome do método especificado no parâmetro Address.

Declaração
```
class function MethodName(Address: Pointer): ShortString;
```

Componentes aos quais se aplica:

Todos os objetos.

Minimize

Descrição

Esse método minimiza a aplicação.

Declaração
```
procedure Minimize;
```

Exemplo

O trecho de código a seguir minimiza a aplicação.
```
Application.Minimize;
```

Componentes aos quais se aplica:
TApplication

MouseToCell

Descrição

Esse método retorna nos parâmetros ARow e ACol a linha e a coluna da célula na qual o mouse está posicionado.

Declaração
```
procedure MouseToCell(X, Y: Integer; var ACol, ARow: Longint);
```

Exemplo

O trecho de código a seguir faz com que, ao se clicar com o mouse sobre uma célula de um componente StringGrid1do tipo TStringGrid, a célula exiba o número da linha e coluna em que se situa.

```
procedure TForm1.StringGrid1MouseUp(Sender: TObject; Button: TMouseButton;
 Shift: TShiftState; X, Y: Integer);
var
      Column, Row: Longint;
begin
      StringGrid1.MouseToCell(X, Y, Column, Row);
      StringGrid1.Cells[Column, Row] := 'Col ' +
      IntToStr(Column) +
        ',Row ' + IntToStr(Row);
end;
```

Componentes aos quais se aplica:
TDrawGrid e TStringGrid

Move

Descrição

Esse método muda a posição de um item em uma lista de objetos ou em uma lista de strings da posição definida pelo parâmetro CurIndex para a posição definida pelo parâmetro NewIndex.

Declaração
```
procedure Move(CurIndex, NewIndex: Integer);
```

Componentes aos quais se aplica:
TList, TStringList e TStrings

MoveBy

Descrição

Esse método move o ponteiro do número de registros definido no parâmetro Distance.

Declaração
```
procedure MoveBy(Distance: Integer);
```

Componentes aos quais se aplica:
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery, TIBStoredProc, TTable, TQuery e TStoredProc

MoveTo

Descrição

Para objetos do tipo TCanvas, esse método muda a posição corrente da caneta para o ponto (X,Y).

Para objetos do tipo TOutlineNode, esse método muda a posição corrente do item dentro do componente TOutline.

Declaração

Para objetos do tipo TCanvas:

```
procedure MoveTo(X, Y: Integer);
```

Para objetos do tipo TOutlineNode:

```
procedure MoveTo(Destination: Longint; AttachMode: TAttachMode);
```

Componentes aos quais se aplica:

```
TCanvas e TOutlineNode
```

Next

Descrição

Para componentes do tipo TForm, esse método torna ativa a próxima janela-filha de uma aplicação MDI.

Para componentes do tipo TMediaPlayer, esse método faz com que o dispositivo multimídia avance uma trilha.

Para componentes derivados da classe TDataset, move o ponteiro para o próximo registro de um banco de dados.

Declaração

```
procedure Next;
```

Exemplo

A linha de código a seguir torna ativa a próxima janela-filha de um formulário Form1 com a propriedade FormStyle igual a fsMDIForm:

```
Form1.Next;
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery, TIBSQL,
TIBStoredProc, TTable, TQuery, TStoredProc,
 Tform, TmediaPlayer, TSimpleDataset, TSQLDataset, TSQLTable, TSQLQuery e TSQLStoredProc
```

NewInstance

Descrição

Esse método aloca memória para uma nova instância de um objeto e retorna um ponteiro para o objeto criado.

Declaração

```
class function NewInstance: TObject; virtual;
```

Componentes aos quais se aplica:

Todos os objetos.

NewPage

Descrição

Para componentes do tipo TQuickReport, esse método força uma quebra de página durante a preparação de um relatório.

Para componentes do tipo TPrinter e TQRPrinter, esse método inicia a impressão de uma nova página.

Declaração

```
procedure NewPage
```

Componentes aos quais se aplica:
```
TQuickReport, TPrinter e TQRPrinter
```

NonExistent

Descrição

Esse método verifica se o objeto associado foi desativado.

Declaração
```
function NonExistent: Boolean;
```

Interfaces às quais se aplica:
```
ICorbaObject
```

NormalizeTopMosts

Descrição

Esse método faz com que os formulários com propriedade FormStyle igual a fsStayOnTop se comportem como formulários comuns (com propriedade FormStyle igual a fsNormal).

Declaração
```
procedure NormalizeTopMosts;
```

Exemplo

O trecho de código a seguir executa o método NormalizeTopMosts em uma aplicação.
```
Application. NormalizeTopMosts;
```

Componentes aos quais se aplica:
```
TApplication
```

OleObjAllocated

Descrição

Esse método verifica se um componente do tipo TOLEContainer possui ou não um objeto OLE.

Declaração
```
function OleObjAllocated: Boolean;
```

Exemplo

O código a seguir exibe uma mensagem se um objeto OLEContainer1 do tipo TOLEContainer contiver um objeto OLE.
```
if OLEContainer1.OleObjAllocated = True then ShowMessage('Contém um objeto OLE');
```

Componentes aos quais se aplica:
```
TOLEContainer
```

Open

Descrição

Para componentes do tipo TMediaPlayer, o método Open faz com que um dispositivo multimídia seja aberto.

Para objetos do tipo TClipboard, o método Open abre o objeto que representa o clipboard e evita que outra aplicação altere o seu conteúdo até ser fechado pelo método Close.

Para componentes do tipo TADOConnection, TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery, TIBStoredProc, TTable, TQuery e TStoredProc, abre o banco de dados associado e o deixa ativo.

Para componentes do tipo TDatabase, esse método realiza a conexão com o servidor.

Para componentes do tipo TRvProject, abre o projeto de relatório representado pelo componente.

Declaração

```
procedure Open;
```

Exemplo

O trecho de código a seguir faz com que um dispositivo multimídia acione seu método

Open com um botão chamado Open do tipo TButton:

```
procedure TForm1.OpenClick(Sender: TObject);
begin
      MediaPlayer1.Open;
end;
```

Componentes aos quais se aplica:

```
TClipBoard, TClientDataset, TDatabase, TMediaPlayer, TADOConnection, TADODataset, TADOTable,
TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery, TIBStoredProc, TTable, TQuery,
TStoredProc, TSimpleDataset, TSQLDataset, TSQLTable, TSQLQuery e TSQLStoredProc.
```

OpenDatabase

Descrição

O método OpenDatabase é usado para abrir um componente do tipo TDatabase, cuja propriedade DatabaseName é igual ao parâmetro DatabaseName, e se não o localizar cria um novo componente.

Declaração

```
function OpenDatabase(const DatabaseName: string): TDatabase;
```

Exemplo

O trecho de código a seguir abre um componente do tipo TDatabase associado a um banco de dados denominado Dados:

```
Session.OpenDatabase('DADOS');
```

Componentes aos quais se aplica:

```
TSession
```

OpenIndexFile

Descrição

Esse método abre um arquivo de índices definido para tabelas no formato dBase.

Declaração

```
procedure OpenIndexFile(const IndexFileName: string);
```

Componentes aos quais se aplica:

```
TTable
```

OpenLink

Descrição

Esse método inicia uma conversação DDE.

Declaração

```
function OpenLink: Boolean;
```

Exemplo

O trecho de código a seguir inicia uma conversação DDE:

```
DDEClientConv1.OpenLink;
```

Componentes aos quais se aplica:
TDDEClientConv

Overlay

Descrição

Esse método transforma a imagem definida pelo parâmetro ImageIndex em uma imagem de superposição, retornando-a no parâmetro Overlay.

Declaração
```
function Overlay(ImageIndex: Integer; Overlay: TOverlay): Boolean;
```

Componentes aos quais se aplica:
TImageList

Pack

Descrição

Esse método deleta os elementos de uma lista cujo ponteiro tem valor nil.

Declaração
```
procedure Pack;
```

Componentes aos quais se aplica:
TList

ParamByName

Descrição

Esse método retorna o elemento da propriedade Params cuja propriedade Name é igual ao parâmetro Value.

Declaração
```
function ParamByName(const Value: string): TParam;
```

Componentes aos quais se aplica:
TIBStoredproc, TStoredProc, TIBQuery e TQuery

PasteFromClipboard

Descrição

Esse método copia o texto do clipboard para o controle na posição atual do cursor, substituindo qualquer texto selecionado no controle.

Declaração
```
procedure PasteFromClipboard;
```

Exemplo

O trecho de código a seguir copia o texto do clipboard para um controle Edit1 do tipo TEdit.
```
Edit1.PasteFromClipboard;
```

Componentes aos quais se aplica:
TDBEdit, TDBImage, TDBMemo, TEdit, TMaskEdit e TMemo

Pause

Descrição

O método Pause provoca uma parada (pausa) em um dispositivo multimídia. Caso o dispositivo já esteja sob uma pausa causada por uma chamada a esse método, o método Resume será acionado.

Declaração

```
procedure Pause;
```

Exemplo

O trecho de código a seguir faz com que um dispositivo multimídia acione seu método Pause com um botão chamado Pause do tipo TButton:

```
procedure TForm1.PauseClick(Sender: TObject);
begin
      MediaPlayer1.Pause;
end;
```

Componentes aos quais se aplica:

```
TMediaPlayer
```

PauseOnly

Descrição

O método PauseOnly é idêntico ao método Pause, exceto por não acionar o método Resume caso o dispositivo já esteja sob uma pausa causada por uma chamada a um método Pause.

Declaração

```
procedure PauseOnly;
```

Exemplo

Veja o método Pause.

Componentes aos quais se aplica:

```
TMediaPlayer
```

Peek

Descrição

Este método retorna um ponteiro para o próximo elemento armazenado na estrutura de dados representada pelo objeto, que pode ser uma Pilha ou uma Fila.

Este método, ao contrário do que ocorre com o método Pop, não remove o ponteiro da estrutura.

Declaração

```
function Peek: Pointer;
```

Componentes aos quais se aplica:

```
TQueue e TStack.
```

Perform

Descrição

Esse método habilita um controle a enviar uma mensagem para si. A mensagem é passada por meio do parâmetro Msg.

Declaração

```
function Perform(Msg, WParam: Word; LParam: Longint): Longint;
```

Componentes aos quais se aplica:

Todos os controles.

Pie

Descrição

Esse método desenha um setor elíptico, sendo a elipse circunscrita pelo retângulo definido pelos pontos (X1, Y1) e (X2,Y2), e limitada pelas linhas radiais que vão do centro da elipse aos pontos (X3,Y3) e (X4,Y4).

Declaração

```
procedure Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Longint);
```

Componentes aos quais se aplica:

```
TCanvas
```

Play

Descrição

O método Play inicia a reprodução em um dispositivo multimídia (para um componente TMediaPlayer) ou de um clip AVI (para um componente TAnimate) – nesse caso, os números que identificam os quadros inicial e final, bem como o número de execuções, são passados como parâmetros.

Declaração

Para componentes TMediaPlayer:

```
procedure Play;
```

Para componentes TAnimate:

```
procedure Play(FromFrame, ToFrame: Word; Count: Integer);
```

Exemplo

O trecho de código a seguir inicia a reprodução em um dispositivo multimídia quando um botão Play do tipo TButton é selecionado:

```
procedure TForm1.PlayClick(Sender: TObject);
begin
      MediaPlayer1.Play;
end;
```

Componentes aos quais se aplica:

```
TAnimate e TMediaPlayer
```

PokeData

Descrição

Esse método envia um dado (definido no parâmetro Data) para um item (definido no parâmetro Item) de uma aplicação servidora em uma conversação DDE.

Declaração

```
function PokeData(Item: string; Data: PChar): Boolean;
```

Componentes aos quais se aplica:

```
TDDEClientConv
```

PokeDataLines

Descrição

Esse método envia um dado (definido como uma lista de strings no parâmetro Data) para um item (definido no parâmetro Item) de uma aplicação servidora em uma conversação.

Declaração

```
function PokeDataLines(Item: string; Data: TStrings): Boolean;
```

Componentes aos quais se aplica:
```
TDDEClientConv
```

Polygon

Descrição

Esse método desenha um polígono cujos vértices estão definidos na array de pontos passada no parâmetro Points e pintado com o pincel corrente.

Declaração
```
procedure Polygon(Points: array of TPoint);
```

Componentes aos quais se aplica:
```
TCanvas
```

Polyline

Descrição

Esse método desenha uma linha poligonal cujos vértices estão definidos na array de pontos passada no parâmetro Points, utilizando-se a caneta corrente.

Declaração
```
procedure Polyline(Points: array of TPoint);
```

Componentes aos quais se aplica:
```
TCanvas
```

Pop

Descrição

Este método retorna um ponteiro para o próximo elemento armazenado na estrutura de dados representada pelo objeto, que pode ser uma Pilha ou uma Fila.

Este método remove o ponteiro da estrutura.

Declaração
```
function Pop: Pointer;
```

Componentes aos quais se aplica:
```
TQueue e TStack.
```

Popup

Descrição

O método Popup exibe um menu flutuante na posição especificada pelos parâmetros X e Y.

Declaração
```
procedure Popup(X, Y: Integer);
```

Componentes aos quais se aplica:
```
TPopup
```

Post

Descrição

Esse método grava o registro corrente no banco de dados.

Declaração
```
procedure Post;
```

Componentes aos quais se aplica:
```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery,  TStoredProc, TClientDataset, TIBClientDataset e TSimpleDataset
```

Prepare

Descrição

Para componentes do tipo TIBSQL, TIBQuery, TIBStoredoroc, TADOQuery, TADOStoredproc, TStoredProc e TQuery, esse método envia uma busca parametrizada para o Borland Database Engine ou outro mecanismo de acesso a banco de dados.

Para componentes do tipo TQuickReport, esse método permite que se crie um relatório sem uma pré-visualização ou impressão.

Declaração
```
procedure Prepare;
```

Componentes aos quais se aplica:
```
TQuickReport, TADOQuery, TADOStoredProc, TIBDataset, TIBQuery, TIBSQL,  TIBStoredproc,
TStoredProc e TQuery
```

Preview

Descrição

Esse método permite a pré-visualização de um relatório.

Declaração
```
procedure Preview
```

Componentes aos quais se aplica:
```
TQuickReport e TQRPrinter
```

Previous

Descrição

Para objetos do tipo TForm, esse método torna ativa a janela-filha anterior de uma aplicação MDI.

Para objetos do tipo TMediaPlayer, faz com que a trilha apontada pelo componente seja a anterior à trilha atual.

Declaração
```
procedure Previous;
```

Exemplo

A linha de código a seguir torna ativa a janela-filha anterior de um formulário Form1 com a propriedade FormStyle igual a fsMDIForm.
```
Form1.Previous;
```

Componentes aos quais se aplica:
```
TForm e TMediaPlayer
```

Print

Descrição

Para componentes do tipo Tform, o método Print faz com que o formulário seja impresso.

Para componentes do tipo TQRPrinter, o método Print imprime um relatório.

Declaração

Para componentes do tipo TForm e TQRPrinter: procedure Print;

Exemplo

A linha de código a seguir faz com que o formulário Form1 seja impresso:

```
Form1.Print;
```

Componentes aos quais se aplica:

```
TForm, TQRPrinter
```

Prior

Descrição

Esse método move o ponteiro para o registro anterior de um banco de dados.

Declaração

```
procedure Prior;
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery, TStoredProc, TClientDataset, TIBClientDataset e TSimpleDataset
```

ProcessMessages

Descrição

Esse método interrompe a execução da aplicação de forma que o Windows possa responder a mensagens pendentes.

Declaração

```
procedure ProcessMessages;
```

Exemplo

O trecho de código a seguir executa o método ProcessMessages em uma aplicação.

```
Application. ProcessMessages;
```

Componentes aos quais se aplica:

```
TApplication
```

Push

Descrição

Este método adiciona um ponteiro para um objeto em uma estrutura de dados representada pelo objeto, que pode ser uma Pilha ou uma Fila.

Declaração

```
procedure Push(AItem: Pointer);
```

Componentes aos quais se aplica:

```
TQueue e TStack.
```

QueueEvents

Descrição

Este método inicia a notificação de eventos para a aplicação.

Declaração

```
procedure QueueEvents;
```

Componentes aos quais se aplica:
```
TIBEvents
```

Read

Descrição

Esse método copia os bytes armazenados em um campo para a variável definida no parâmetro Buffer.

Declaração
```
function Read(var Buffer; Count: Longint): Longint;
```

Componentes aos quais se aplica:
```
TBlobStream
```

ReadBool

Descrição

Esse método lê um valor booleano (definido no parâmetro Ident) de uma seção (definida no parâmetro Section) de um arquivo INI.

Declaração
```
function ReadBool(const Section, Ident: string; Default: Boolean): Boolean;
```

Componentes aos quais se aplica:
```
TIniFile
```

ReadInteger

Descrição

Esse método lê um valor inteiro (definido no parâmetro Ident) de uma seção (definida no parâmetro Section) de um arquivo INI.

Declaração
```
function ReadInteger(const Section, Ident: string; Default: Longint): Longint;
```

Componentes aos quais se aplica:
```
TIniFile
```

ReadSection

Descrição

Esse método lê todas as variáveis de uma seção (definida no parâmetro Section) de um arquivo INI e as armazena na lista de strings definida pelo parâmetro Strings.

Declaração
```
procedure ReadSection (const Section: string; Strings: TStrings);
```

Componentes aos quais se aplica:
```
TIniFile
```

ReadSectionValues

Descrição

Esse método lê todas as variáveis e seus valores de uma seção (definida no parâmetro Section) de um arquivo INI e os armazena na lista de strings definida pelo parâmetro
```
Strings.
```

Declaração
```
procedure ReadSectionValues(const Section: string; Strings: TStrings);
```

Componentes aos quais se aplica:
TIniFile

ReadString

Descrição

Esse método lê o valor de uma string (definida no parâmetro Ident) de uma seção (definida no parâmetro Section) de um arquivo INI.

Declaração
```
function ReadString(const Section, Ident, Default: string): string;
```

Componentes aos quais se aplica:
TIniFile

Realign

Descrição

Esse método força um realinhamento dos controles-filhos do controle corrente.

Declaração
```
procedure Realign;
```

Componentes aos quais se aplica:

Todos os controles.

Rectangle

Descrição

Esse método desenha um retângulo definido pelos pontos (X1,Y1) – vértice superior
esquerdo e (X2,Y2) – vértice inferior direito, com o pincel e a caneta corrente.

Declaração
```
procedure Rectangle(X1, Y1, X2, Y2: Integer);
```

Componentes aos quais se aplica:
TCanvas

Refresh

Descrição

No caso de componentes TTable, TQuery e TStoredProc, esse método garante a atualização do conjunto de dados exibidos. Para os demais controles, o método Refresh atualiza as imagens exibidas pelos controles, chamando internamente os métodos Invalidate (que apaga o conteúdo atual) e Update.

Declaração
```
procedure Refresh;
```

Exemplo

O trecho de código a seguir força a atualização da imagem exibida em um controle Image1 do tipo TImage ao se clicar sobre um botão chamado Button1 com o botão esquerdo do mouse.

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
       Image1.Refresh;
end;
```

Componentes aos quais se aplica:

Todos os controles e os componentes derivados de TDataset.

RegisterChanges

Descrição

Esse método é chamado por um objeto que deseja ser informado de alterações ocorridas na lista de imagens.

Declaração

```
procedure RegisterChanges(Value: TChangeLink);
```

Componentes aos quais se aplica:

```
TListImages
```

RegisterEvents

Descrição

Este método registra, para a aplicação, os eventos definidos pela propriedade Events do componente.

Declaração

```
procedure RegisterEvents;
```

Componentes aos quais se aplica:

```
TIBEvents
```

Release

Descrição

Esse método é idêntico ao método Free, exceto que aguarda que a execução de código de todos os eventos pendentes do formulário ou componentes nele inseridos seja finalizada.

Declaração

```
procedure Release;
```

Exemplo

A linha de código a seguir mostra um formulário chamado Form1 executando o seu método Release.

```
Form1.Release;
```

Componentes aos quais se aplica:

```
TForm
```

ReleaseHandle

Descrição

Esse método libera o handle usado pelo objeto.

Declaração

```
function ReleaseHandle: HBitmap;
```

Exemplo

A linha de código a seguir libera o handle de um objeto Bitmap1 do tipo TBitmap.

```
Bitmap1.ReleaseHandle;
```

Componentes aos quais se aplica:

```
TBitmap
```

ReleasePalette

Descrição

Esse método libera o handle da paleta de cores usada pelo objeto.

Declaração

```
function ReleasePalette: HPalette;
```

Exemplo

A linha de código a seguir libera o handle da paleta de cores de um objeto Bitmap1 do tipo TBitmap.

```
Bitmap1.ReleasePalette;
```

Componentes aos quais se aplica:

```
TBitmap
```

Remove

Descrição

Para componentes dos tipos TList e TMenuItem, esse método remove o item especificado pelo parâmetro Item e retorna o valor da posição que era ocupada pelo Item.

Declaração

```
function Remove(Item: Pointer): Integer;
```

Componentes aos quais se aplica:

```
Tlist e TMenuItem
```

RemoveAllPassword

Descrição

Esse método, que só se aplica a componentes relacionados a bancos de dados do tipo Paradox, desconsidera todas as senhas fornecidas pelo usuário, solicitando nova senha para que as tabelas possam ser reabertas.

Declaração

```
procedure RemoveAllPasswords;
```

Exemplo

O trecho de código a seguir aciona o método RemoveAllPassword:

```
Session.RemoveAllPassword;
```

Componentes aos quais se aplica:

```
TSession
```

RemoveAllSeries

Descrição

Esse método remove todas as séries do gráfico exibido no componente.

Declaração

```
procedure RemoveAllSeries;
```

Componentes aos quais se aplica:

```
TChart e TDBChart
```

RemoveComponent

Descrição

Esse método remove um componente passado no parâmetro AComponent na lista de componentes definida pela propriedade Components do componente atual.

Declaração

```
procedure RemoveComponent(AComponent: Tcomponent);
```

Exemplo

O trecho de código a seguir remove um botão Button1 do tipo TButton na array de componentes de um formulário chamado Form1.

```
Form1.RemoveComponent(Button1);
```

Componentes aos quais se aplica:

Todos os componentes.

RemoveDatabase

Descrição

Este método remove da sua propriedade Databases o objeto da classe TIBDatabase cujo índice é passado como parâmetro.

Declaração

```
procedure RemoveDatabase(Idx: Integer);
```

Componentes aos quais se aplica:

```
TIBTransaction
```

RemoveDatabases

Descrição

Este método remove da sua propriedade Databases todos os objetos da classe TIBDatabase por ela referenciados.

Declaração

```
procedure RemoveDatabases;
```

Componentes aos quais se aplica:

```
TIBTransaction
```

RemovePassword

Descrição

Esse método, que só se aplica a componentes relacionados a bancos de dados do tipo Paradox, elimina a senha definida no parâmetro Password do conjunto de senhas autorizadas.

Declaração

```
procedure RemovePassword(const Password: string);
```

Exemplo

O trecho de código a seguir aciona o método RemovePassword para eliminar uma senha:

```
Session.RemovePassword('Senha');
```

Componentes aos quais se aplica:

```
TSession
```

RemoveSeries

Descrição

Esse método remove uma série do gráfico exibido no componente.

Declaração

```
procedure RemoveSeries(ASeries : TChartSeries);
```

Componentes aos quais se aplica:

```
TChart e TDBChart
```

RemoveTransaction

Descrição

Esse método cancela a associação da transação cujo índice é passado como parâmetro.

Declaração

```
procedure RemoveTransaction(Idx: Integer);
```

Componentes aos quais se aplica:

```
TIBDatabase
```

RemoveTransactions

Descrição

Esse método cancela a associação de todas as transações.

Declaração

```
procedure RemoveTransactions;
```

Componentes aos quais se aplica:

```
TIBDatabase
```

Repaint

Descrição

O método Repaint atualiza as imagens exibidas pelos controles sem apagar a imagem atual.

Declaração

```
procedure Repaint;
```

Exemplo

O trecho de código a seguir força a atualização da imagem exibida em um controle chamado Image1 do tipo TImage ao se clicar sobre um botão chamado Button1 com o botão esquerdo do mouse.

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
       Image1.Repaint;
end;
```

Observação: Ao contrário do que ocorre com o método Refresh, a imagem atual não é apagada.

Componentes aos quais se aplica:

Todos os controles.

ReplaceIcon

Descrição

Esse método substitui a imagem armazenada na posição definida pelo parâmetro Index pelo ícone passado pelo parâmetro Image.

Declaração

```
procedure ReplaceIcon(Index: Integer; Image: TIcon);
```

Componentes aos quais se aplica:

```
TListImages
```

ReplaceMasked

Descrição

Esse método substitui a imagem armazenada na posição definida pelo parâmetro Index pela imagem passada no parâmetro NewImage e com a máscara transparente definida na propriedade MaskColor.

Declaração

```
procedure ReplaceMasked(Index: Integer; NewImage: TBitmap; MaskColor: TColor);
```

Componentes aos quais se aplica:

```
TListImages
```

REQUESTDATA

Descrição

Esse método obtém um dado (definido no parâmetro Item) de uma aplicação servidora em uma conversação DDE.

Declaração

```
function RequestData(const Item: string): PChar;
```

Componentes aos quais se aplica:

```
TDDEClientConv
```

RESET

Descrição

Esse método redefine os valores default para as propriedades StartFrame e StopFrame do componente, exibe o seu primeiro quadro e atribui o valor False à sua propriedade Active.

Declaração

```
Procedure Reset;
```

Componentes aos quais se aplica:

```
TAnimate
```

RESOURCELOAD

Descrição

Esse método carrega um recurso do tipo TResType na lista de imagens do componente, com a máscara transparente definida na propriedade MaskColor.

Declaração

```
function ResourceLoad(ResType: TResType; Name: string; MaskColor: TColor): Boolean;
```

Componentes aos quais se aplica:

```
TListImages
```

RESTORE

Descrição

Esse método restaura o tamanho que o formulário principal de uma aplicação tinha antes de ser minimizado ou maximizado.

Declaração

```
procedure Restore;
```

Exemplo

O trecho de código a seguir executa o método Restore em uma aplicação.

```
Application. Restore;
```

Componentes aos quais se aplica:

```
TApplication
```

RestoreTopMosts

Descrição

Esse método faz com que os formulários recuperem seu comportamento definido pela propriedade FormStyle igual a fsStayOnTop após terem seu comportamento alterado pela execução do método NormalizeTopMosts.

Declaração

```
procedure RestoreTopMosts;
```

Exemplo

O trecho de código a seguir executa o método RestoreTopMosts em uma aplicação.

```
Application. RestoreTopMosts;
```

Componentes aos quais se aplica:

```
TApplication
```

Resume

Descrição

Para objetos da classe TMediaPlayer, reinicia a reprodução em um dispositivo multimídia interrompida pelo método Pause.

Para objetos da classe Tthread, reinicia a sua execução.

Declaração

```
procedure Resume;
```

Exemplo

O trecho de código a seguir reinicia a reprodução em um dispositivo multimídia interrompida com um método Pause.

```
MediaPlayer1.Resume;
```

Componentes aos quais se aplica:

```
TMediaPlayer e TThread
```

Rewind

Descrição

O método Rewind define a posição corrente em um dispositivo multimídia como a definida na propriedade Start.

Declaração

```
procedure Rewind;
```

Exemplo

O trecho de código a seguir começa a reprodução no início de um dispositivo multimídia quando um botão chamado Rewind do tipo TButton é selecionado:

```
procedure TForm1.RewindClick(Sender: TObject);
begin
      MediaPlayer1.Rewind;
     MediaPlayer1.Play;
end;
```

Componentes aos quais se aplica:

```
TMediaPlayer
```

RollBack

Descrição

Esse método desfaz a transação corrente e todas as modificações feitas nos registros do banco de dados associado desde a última chamada ao método StartTransaction.

Declaração

```
procedure RollBack;
```

Componentes aos quais se aplica:

```
TDatabase, TIBTransaction
```

RollBackRetaining

Descrição

Esse método desfaz a transação corrente e todas as modificações feitas nos registros do banco de dados associado desde a última chamada ao método StartTransaction, mantendo o contexto de transação corrente.

Declaração

```
procedure RollBackRetaining;
```

Componentes aos quais se aplica:

```
TDatabase, TIBTransaction
```

RoundRect

Descrição

Esse método desenha um retângulo definido pelos pontos (X1,Y1) – vértice superior esquerdo e (X2,Y2) – vértice inferior direito, com os cantos arredondados por um quadrante elíptico de dimensões X3 e Y3, usando o pincel e a caneta correntes.

Declaração

```
procedure RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer);
```

Componentes aos quais se aplica:

```
TCanvas
```

Run

Descrição

Para componentes do tipo TApplication, esse método executa a aplicação. Normalmente, a chamada a esse método é feita no arquivo de projeto (*.DPR) e é incluída automaticamente pelo Delphi.

Declaração

```
procedure Run;
```

Exemplo

O trecho de código a seguir executa uma aplicação:

```
Application.Run
```

Componentes aos quais se aplica:

```
TApplication
```

Save

Descrição

Para componentes do tipo TMediaPlayer, esse método salva o registro corrente em um dispositivo multimídia, em um arquivo definido pela propriedade FileName.

Para componentes do tipo TQRPrinter, esse método salva o relatório corrente em um arquivo, definido pelo parâmetro Filename.

Declaração

Para componentes do tipo TMediaPlayer:

```
procedure Save;
```

Para componentes do tipo TQRPrinter:

```
procedure Save(Filename : String)
```

Exemplo

O trecho de código a seguir inicia a gravação em um dispositivo multimídia quando um botão Save do tipo TButton é selecionado:

```
procedure TForm1.SaveClick(Sender: TObject);
begin
     MediaPlayer1.Save;
end;
```

Componentes aos quais se aplica:

```
TMediaPlayer e TQRPrinter
```

SaveToBitmapFile

Descrição

Esse método grava, em um arquivo no formato bitmap, o gráfico exibido no componente.

Declaração

```
procedure SaveToBitmapFile( Const FileName : String ) ;
```

Componentes aos quais se aplica:

```
TChart e TDBChart
```

SaveToFile

Descrição

O método SaveToFile salva um objeto ou relatório no arquivo definido pelo parâmetro FileName.

Declaração

```
procedure SaveToFile(const FileName: string);
```

Exemplo

A linha de código a seguir faz com que a lista de strings, armazenada na propriedade Lines de um componente Memo1 do tipo TMemo, seja salva em um arquivo chamado temp.txt:

```
Memo1.Lines.SaveToFile('TEMP.TXT');
```

Componentes aos quais se aplica:

```
TBitmap, TBlobField, TClientDataset, TGraphic, TGraphicField, TIcon, TMemoField, TMetaFile,
TPicture, TStringList, TStrings, TOLEContainer, TOutline e TrvProject.
```

SaveToMetaFile

Descrição

Esse método grava, em um arquivo no formato windows metafile (wmf), o gráfico exibido no componente.

Declaração

```
procedure SaveToMetafile(Const FileName : String);
```

Componentes aos quais se aplica:
TChart e TDBChart

SaveToMetaFileEnh

Descrição

Esse método grava, em um arquivo no formato enhanced windows metafile, o gráfico exibido no componente.

Declaração
```
procedure SaveToMetafileEnh(Const FileName : String);
```

Componentes aos quais se aplica:
TChart e TDBChart

SaveToStream

Descrição

Esse método copia o conteúdo do campo ou relatório na stream especificada no parâmetro Stream.

Declaração
```
procedure SaveToStream(Stream: TStream);
```

Exemplo

O trecho de código a seguir salva o conteúdo do campo BlobField1 do tipo TBlobField em uma stream chamada Stream1.
```
BlobField1.SaveToStream(Stream1);
```

Componentes aos quais se aplica:
TClientDataset, TBlobField, TDBMemoBuf, TGraphicField, TmemoBuf, TmemoField e TrvProject.

ScaleBy

Descrição

Esse método faz uma transformação de escala nas dimensões de um componente, em relação às suas dimensões iniciais. Para isso, devem ser fornecidos os parâmetros inteiros M e D que são, respectivamente, o Numerador e o Denominador do fator de escala.

Declaração
```
procedure ScaleBy(M, D: Integer);
```

Exemplo

Se você quiser que um botão Button1 dobre as suas dimensões quando o usuário der um clique sobre ele com o botão do mouse, inclua a seguinte linha de código no seu evento OnClick:
```
Button1.ScaleBy(2,1);
```

Componentes aos quais se aplica:

Todos os controles.

ScreenToClient

Descrição

Esse método faz a transformação das coordenadas de um ponto do sistema de coordenadas da tela para o sistema de coordenadas da área-cliente (o inverso do método ClientToScreen).

Declaração
```
function ScreenToClient (Point: TPoint): TPoint;
```

Exemplo

O trecho de código que se segue define P e Q como variáveis do tipo TPoint no evento OnMouseDown de um formulário. Ao ponto P são atribuídas as coordenadas do ponto em que o botão do mouse foi pressionado (sistema de coordenadas da tela), e esses valores são armazenados no ponto Q após a transformação para o sistema de coordenadas da área-cliente – o formulário.

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
 Shift: TShiftState; X, Y: Integer);
var
 P, Q: TPoint;
begin
 P.X:= Left + X;
 P.Y:= Top + Y;
 Q:= ScreenToClient(P);
end;
```

Esse método é uma função e, conseqüentemente, o seu valor de retorno pode ser atribuído a uma variável.

Componentes aos quais se aplica:

Todos os controles.

ScrollBy

Descrição

O método ScrollBy faz a rolagem do conteúdo de um formulário ou controle-pai. Os parâmetros DeltaX e DeltaY definem a variação, em pixels, ao longo dos eixos X e Y, respectivamente. Um valor positivo de DeltaX provoca uma rolagem para a direita e um valor negativo provoca uma rolagem para a esquerda. Um valor positivo de DeltaY provoca uma rolagem para baixo e um valor negativo provoca uma rolagem para cima.

Declaração

```
procedure ScrollBy(DeltaX, DeltaY: Integer);
```

Exemplo

Se você quiser que o conteúdo de um formulário Form1 role 10 unidades para a direita e para baixo cada vez que o usuário clicar sobre um botão chamado Button1, defina o evento OnClick de Button1 da seguinte forma:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
         ScrollBy(10,10);
end;
```

Componentes aos quais se aplica:

Todos os controles.

ScrollInView

Descrição

Esse método rola o conteúdo de um componente do tipo TForm ou TScrollBox de forma que o controle especificado no parâmetro AControl seja ao menos parcialmente visível.

Declaração

```
procedure ScrollInView(AControl: TControl);
```

Exemplo

A linha de código a seguir rola o conteúdo de um formulário chamado Form1 até que um botão chamado Button1 do tipo TButton esteja ao menos parcialmente visível.

```
ScrollInView(Button1);
```

Componentes aos quais se aplica:

```
TForm e TScrollBox
```

Seek

Descrição

Para objetos da classe TBlobStream, esse método reposiciona o ponteiro de leitura em um campo de um dado.

Para objetos da classe TAnimate, esse método reposiciona o clipe no quadro cujo índice é passado como parâmetro.

Declaração

Para objetos da classe TBlobStream:

```
function Seek(Offset: Longint; Origin: Word): Longint;
```

Para objetos da classe TAnimate:

```
procedure Seek(Frame: SmallInt);
```

Componentes aos quais se aplica:

```
TAnimate, TBlobStream
```

SelectAll

Descrição

O método SelectAll seleciona todos os itens de um componente.

Declaração

```
procedure SelectAll;
```

Exemplo

Se você quiser selecionar todos os itens de um componente Memo1 do tipo TMemo, basta incluir a seguinte linha de código:

```
Memo1.SelectAll;
```

Componentes aos quais se aplica:

```
TComboBox, TDBComboBox, TDBEdit, TDBMemo, TDriveComboBox, TEdit, TFilterComboBox, TMaskEdit e
TMemo
```

SelectNext

Descrição

O método SelectNext seleciona a próxima guia em um controle do tipo TTabSet. O parâmetro booleano Direction define se a próxima guia a ser selecionada deve ser a da direita (True) ou a da esquerda (False).

Declaração

```
procedure SelectNext(Direction: Boolean);
```

Exemplo

A linha de código a seguir seleciona a guia da esquerda da guia correntemente selecionada.

```
TabSet1.SelectNext(False);
```

Componentes aos quais se aplica:

```
TTabSet
```

SelectNextPage

Descrição

Esse método permite que se especifique a próxima página a ser exibida pelo controle. Se o parâmetro GoForWard for igual a True, a próxima página será a numericamente subseqüente. Se o parâmetro GoForWard for igual a False, a próxima página será a numericamente anterior.

Declaração

```
procedure SelectNextPage(GoForward: Boolean);
```

Componentes aos quais se aplica:

```
TPageControl
```

SendToBack

Descrição

O método SendToBack coloca o controle atrás de todos os controles que existem no mesmo formulário (é o inverso do método BringToFront).

Declaração

```
procedure SendToBack;
```

Exemplo

O trecho de código a seguir faz com que um botão de rádio chamado RadioButton1 seja colocado atrás de todos os outros componentes que existem no mesmo formulário:

```
RadioButton1.Bring SendToBack;
```

Componentes aos quais se aplica:

Todos os controles e componentes do tipo TForm.

SeriesCount

Descrição

Esse método retorna o número de séries mostradas no gráfico exibido no componente.

Declaração

```
function SeriesCount : Longint ;
```

Componentes aos quais se aplica:

```
TChart e TDBChart
```

SetAsHandle

Descrição

Esse método define um handle para um objeto com o formato definido no parâmetro Format a ser armazenado no clipboard.

Declaração

```
function SetAsHandle (Format: Word): THandle;
```

Componentes aos quais se aplica:
```
TClipboard
```

SetBounds

Descrição

O método SetBounds define de uma só vez o valor das variáveis Left, Top, Width e Height de um componente, por meio dos valores passados pelos parâmetros ALeft, ATop, AWidth e Aheight, respectivamente.

Declaração
```
procedure Setbounds(ALeft, ATop, AWidth, AHeight: Integer);
```

Exemplo

O trecho de código a seguir dobra o tamanho de um botão chamado Button1 quando o usuário clica sobre ele com o botão esquerdo do mouse:
```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Button1.SetBounds(Left, Top, Width * 2, Height * 2);
end;
```

Componentes aos quais se aplica:

Todos os controles.

SetComponent

Descrição

Esse método copia um componente (definido pelo parâmetro Component) no clipboard.

Declaração
```
procedure SetComponent(Component: TComponent);
```

Componentes aos quais se aplica:
```
TClipboard
```

SetData

Descrição

Esse método armazena no campo o valor do parâmetro Buffer.

Declaração
```
procedure SetData(Buffer: Pointer);
```

Componentes aos quais se aplica:
```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

SetFields

Descrição

Esse método atribui aos campos de um banco de dados os valores definidos na array passado pelo parâmetro Values.

Declaração
```
procedure SetFields(const Values: array of const);
```

Componentes aos quais se aplica:
```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery e TStoredProc
```

SetFocus

Descrição

O método SetFocus faz com que o controle receba o foco da aplicação.

Declaração

```
procedure SetFocus;
```

Exemplo

O trecho de código a seguir faz com que um controle Edit1 do tipo TEdit receba o foco da aplicação:

```
Edit1.SetFocus;
```

Componentes aos quais se aplica:

Todos os controles.

SetKey

Descrição

Esse método é usado para definir valores para o buffer de pesquisa em uma tabela.

Declaração

```
procedure SetKey;
```

Componentes aos quais se aplica:

```
TClientDataset, TIBClientDataset, TSimpleDataset e TTable
```

SetLink

Descrição

Esse método define a aplicação servidora e o tópico em uma conversação DDE.

Declaração

```
function SetLink(Service: string; Topic: string): Boolean;
```

Componentes aos quais se aplica:

```
TDDEClientConv
```

SetParams

Descrição

O método SetParams define com os parâmetros Aposition, AMin e AMax os valores das propriedades Position, Min e Max de uma barra de rolamento.

Declaração

```
procedure SetParams(APosition, AMin, AMax: Integer);
```

Exemplo

Você pode definir os parâmetros de uma barra de rolamento com uma chamada ao método SetParams, como no trecho de código a seguir:

```
ScrollBar1.SetParams(10,0,100);
```

Componentes aos quais se aplica:

```
TScrollBar
```

SetPrincipal

Descrição

Esse método envia uma array de bytes para a aplicação servidora.

Declaração

```
procedure SetPrincipal(const Principal: TCorbaPrincipal);
```

onde TCorbaPrincipal é definido como:

```
TCorbaPrincipal = array of Byte;
```

Interfaces às quais se aplica:

```
ICorbaObject
```

SETPRINTER

Descrição

Esse método especifica uma impressora como a impressora corrente.

Declaração

```
procedure SetPrinter(ADevvice, ADriver, APort: PChar; ADeviceMode: THandle);
```

Componentes aos quais se aplica:

```
TPrinter
```

SETRANGE

Descrição

Esse método corresponde a chamar os métodos SetRangeStart, SetRangeEnd e

```
ApplyRange.
```

Declaração

```
procedure SetRange(const StartValues, EndValues: array of const);
```

Componentes aos quais se aplica:

```
TClientDataset, TIBClientDataset, TSimpleDataset e TTable
```

SETRANGEEND

Descrição

Esse método indica que as atribuições de campo que se seguirem definirão o término da faixa de registros a incluir no banco de dados associado.

Declaração

```
procedure SetRangeEnd;
```

Componentes aos quais se aplica:

```
TClientDataset, TTable
```

SETRANGESTART

Descrição

Esse método indica que as atribuições de campo que se seguirem definirão o início da faixa de registros a incluir no banco de dados associado.

Declaração

```
procedure SetRangeStart;
```

Componentes aos quais se aplica:

```
TClientDataset, TTable
```

SetSelTextBuf

Descrição

O método SetSelTextBuf atribui, ao texto selecionado em um controle, a string terminada em zero apontada pela variável Buffer.

Declaração

```
procedure SetSelTextBuf(Buffer: PChar);
```

Exemplo

Se você quiser que um componente Edit1 do tipo TEdit substitua o texto selecionado pela mensagem 'Você pressionou o botão 1' quando o usuário clicar sobre um botão chamado Button1, pode incluir a seguinte linha de código no evento OnClick associado ao botão:

```
Edit1.SetSelTextBuf('Você pressionou o botão 1');
```

Componentes aos quais se aplica:

```
TComboBox, TDBComboBox, TDBEdit, TDBMemo, TEdit, TMaskEdit e TMemo
```

SetTabFocus

Descrição

O método SetTabFocus faz com que a página ativa seja aquela cujo valor da propriedade PageIndex é igual ao parâmetro Index.

Declaração

```
procedure SetTabFocus(Index: Integer);
```

Exemplo

O trecho de código a seguir faz com que a segunda página de um componente TabbedNotebook1 do tipo TTabbedNotebook seja a página ativa.

```
TabbedNotebook1.SetTabFocus[1];
```

Componentes aos quais se aplica:

```
TTabbedNotebook
```

SetText

Descrição

Esse método armazena uma lista de strings, definida como uma string de terminação nula no parâmetro Text.

Declaração

```
procedure SetText(Text: PChar);
```

Componentes aos quais se aplica:

```
Tstrings e TStringList
```

SetTextBuf

Descrição

O método SetTextBuf atribui ao texto exibido por um controle a string terminada em zero apontada pela variável Buffer.

Declaração

```
procedure SetTextBuf(Buffer: PChar);
```

Exemplo

Se você quiser que um componente Edit1 do tipo TEdit exiba a mensagem 'Você pressionou o botão 1' quando o usuário clicar sobre um botão chamado Button1, pode incluir a seguinte linha de código no evento OnClick associado ao botão:

```
Edit1.SetTextBuf('Você pressionou o botão 1');
```

Componentes aos quais se aplica:

Todos os controles e objetos do tipo TClipboard.

SetUpdateState

Descrição

O método SetUpdateState define se a reindexação dos itens em um componente do tipo TOutline deve ou não ser feita automaticamente, de acordo com o valor passado pelo parâmetro Value.

Declaração

```
procedure SetUpdateState(Value: Boolean);
```

Exemplo

Você pode acionar o método SetUpdateState de um componente Outline1 do tipo TOutline mediante a inclusão de uma linha de código como:

```
Outline1.SetUpdateState(True);
```

Componentes aos quais se aplica:

```
TOutline;
```

Show

Descrição

O método Show torna visível um componente, fazendo sua propriedade Visible igual a True (é o inverso do método Hide).

Declaração

```
procedure Show;
```

Exemplo

Se você quiser que um botão chamado Button1 apareça em um formulário que o contém (Form1) ao se clicar sobre o formulário com o botão esquerdo do mouse, defina o evento OnClick do formulário da seguinte forma:

```
procedure TForm1.FormClick(Sender: TObject);
begin
       Button1.Show;
end;
```

Componentes aos quais se aplica:

Todos os controles e componentes do tipo TForm.

ShowCubeDialog

Descrição

Esse método exibe a caixa de diálogo do editor de propriedades do componente.

Declaração

```
procedure ShowCubeDialog;
```

Componentes aos quais se aplica:
```
TDecisionCube
```

ShowDragImage

Descrição

Esse método exibe a imagem correntemente arrastada, que havia sido ocultada por uma chamada ao método HideDragImage.

Declaração
```
procedure ShowDragImage;
```

Componentes aos quais se aplica:
```
TImageList
```

ShowException

Descrição

Esse método exibe uma mensagem informando uma exceção ocorrida durante a execução da aplicação.

Declaração
```
procedure ShowException(E: Exception);
```

Exemplo

O trecho de código a seguir executa o método ShowException em uma aplicação:
```
var
     E : Exception;
begin
     E := Exception.Create('Exceção');
     Application.ShowException(E);
end;
```

Componentes aos quais se aplica:
```
Tapplication
```

ShowModal

Descrição

O método ShowModal exibe um formulário de forma modal, isto é, a aplicação só pode continuar fora do formulário depois que este é fechado pelo usuário. Quando isso ocorre, a propriedade ModalResult do formulário retorna um valor diferente de zero.

Declaração
```
procedure ShowModal;
```

Exemplo

Se você quiser que um formulário chamado Form2 apareça de forma modal ao se clicar sobre um botão Button1 de um formulário chamado Form1, defina o evento OnClick de Button1 da seguinte forma:
```
procedure TForm1.Button1Click(Sender: TObject);
begin
       Form2.ShowModal;
end;
```

Componentes aos quais se aplica:
```
TForm
```

StartRecording

Descrição

O método StartRecording inicia a gravação no registro corrente ou na posição definida pela propriedade StartPos em um dispositivo multimídia.

Declaração

```
procedure StartRecording;
```

Exemplo

O trecho de código a seguir executa o método StartRecording em um dispositivo multimídia:

```
MediaPlayer1.StartRecording;
```

Componentes aos quais se aplica:

```
TMediaPlayer
```

StartTransaction

Descrição

Esse método inicia uma transação no nível de isolamento definido pela propriedade Transisolation.

Declaração

```
procedure StartTransaction;
```

Componentes aos quais se aplica:

```
TDatabase, TIBTransaction
```

Step

Descrição

O método Step faz com que o dispositivo multimídia avance um certo número de quadros, especificado na propriedade Frames.

Declaração

```
procedure Step;
```

Exemplo

O trecho de código a seguir faz com que um dispositivo multimídia acione seu método Step com um botão Step do tipo TButton:

```
procedure TForm1.StepClick(Sender: TObject);
begin
      MediaPlayer1.Step;
end;
```

Componentes aos quais se aplica:

```
TMediaPlayer
```

Stop

Descrição

O método Stop interrompe a reprodução ou a gravação em um dispositivo multimídia (no caso do componente TMediaPlayer) ou do clipe AVI (no caso do componente TAnimate).

Declaração

```
procedure Stop;
```

Exemplo

O trecho de código a seguir interrompe a reprodução ou gravação em um dispositivo multimídia quando um botão Stop do tipo TButton é selecionado:

```
procedure TForm1.StopClick(Sender: TObject);
begin
      MediaPlayer1.Stop;
end;
```

Componentes aos quais se aplica:
TAnimate e TMediaPlayer

StretchDraw

Descrição

Esse método desenha o objeto gráfico definido no parâmetro Graphic na área retangular definida pelo parâmetro Rect, redimensionando o gráfico de forma a ocupar toda a área retangular.

Declaração
```
procedure StretchDraw(const Rect: TRect; Graphic: TGraphic);
```

Componentes aos quais se aplica:
TCanvas

Suspend

Descrição

Esse método suspende a execução de uma thread.

Declaração
```
procedure Suspend;
```

Componentes aos quais se aplica:
TThread

Synchronize

Descrição

Esse método sincroniza a execução do método passado como parâmetro com a thread principal da VCL.

Declaração
```
procedure Synchronize(Method: TThreadMethod);
```

Componentes aos quais se aplica:
TThread

Terminate

Descrição

Para objetos da classe TApplication, esse método finaliza a aplicação.

Para objetos da classe TThread, esse método finaliza a sua execução.

Declaração

Para TApplication e TThread:
```
procedure Terminate;
```

Para TIWApplication
```
procedure Terminate(const AMsg: string);
```

Exemplo

O trecho de código finaliza a execução de uma aplicação:
```
Application.Terminate;
```

Componentes aos quais se aplica:
```
TApplication, TIWApplication e TThread
```

TestConnected

Descrição

Esse método verifica se a conexão ao banco de dados representado pelo componente está ativa.

Declaração
```
function TestConnected: Boolean;
```

Componentes aos quais se aplica:
```
TIBDatabase
```

TextHeight

Descrição

Esse método retorna a altura, em pixels, da string passada no parâmetro text para a fonte corrente.

Declaração
```
function TextHeight(const Text: string): Integer;
```

Componentes aos quais se aplica:
```
TCanvas
```

TextOut

Descrição

Esse método desenha a string passada no parâmetro Text para a fonte corrente na posição X,Y (canto superior esquerdo da string).

Declaração
```
procedure TextOut(X, Y: Integer; const Text: string);
```

Componentes aos quais se aplica:
```
TCanvas
```

TextRect

Descrição

Esse método desenha a string passada no parâmetro Text para a fonte corrente na posição X,Y (canto superior esquerdo da string), em um retângulo de clipping definido no parâmetro Rect.

Declaração
```
procedure TextRect(Rect: TRect; X, Y: Integer; const Text: string);
```

Componentes aos quais se aplica:
```
TCanvas
```

TextWidth

Descrição

Esse método retorna a largura, em pixels, da string passada no parâmetro text para a fonte corrente.

Declaração
```
function TextWidth(const Text: string): Integer;
```

Componentes aos quais se aplica:
```
TCanvas
```

Tile

Descrição

O método Tile exibe todos os formulários-filhos ativos de um formulário-pai com o mesmo tamanho e ocupando toda a área-cliente do formulário-pai. Esse método só se aplica aos formulários-pais cuja propriedade FormStyle vale fsMDIForm. A forma de exibição depende do valor da propriedade TileMode.

Declaração

```
procedure Tile;
```

Exemplo

Crie um item de menu chamado TileFormsClick no menu principal do formulário-pai MDI e defina o seu evento OnClick da seguinte forma, para que os formulários-filhos sejam exibidos lado a lado e verticalmente:

```
procedure TForm1.TileForms1Click(Sender: TObject);
begin
    TileMode := tbVertical;
    Tile;
end;
```

Componentes aos quais se aplica:

```
TForm
```

Truncate

Descrição

Esse método trunca o conteúdo do campo definido pelo objeto na posição corrente do ponteiro, descartando os dados restantes.

Declaração

```
procedure Truncate;
```

Componentes aos quais se aplica:

```
TBlobStream
```

UnLockTable

Descrição

Esse método remove uma restrição imposta a outras aplicações que acessam a tabela para leitura e/ou gravação, feita por uma chamada ao método LockTable.

Declaração

```
procedure LockTable(LockType: TLockType);
```

onde TLockType é um tipo enumerado definido da seguinte maneira:

```
type TLockType = (ltReadLock, ltWriteLock);
```

Componentes aos quais se aplica:

```
TTable
```

UnMerge

Descrição

Esse método desfaz a mesclagem de dois menus de formulários diversos em uma aplicação não-MDI. O parâmetro Menu define o menu a ser retirado da combinação.

Declaração

```
procedure Unmerge(Menu: TMainMenu);
```

Componentes aos quais se aplica:
TMainMenu

UnPrepare

Descrição

Para componentes do tipo TIBDataset, TADOQuery, TIBQuery, TQuery, esse método atribui o valor False à propriedade Prepared do componente.

Para componentes dos tipos TADOStoredProc, TIBStoredproc, e TStoredProc, esse método informa ao servidor que o procedimento armazenado não será mais usado, possibilitando a liberação de recursos do sistema.

Declaração
```
procedure UnPrepare;
```

Componentes aos quais se aplica:
TADOQuery, TADOStoredProc, TIBDataset, TIBQuery, TIBStoredproc, TStoredProc e TQuery

UnRegisterChanges

Descrição

Esse método cancela os efeitos da chamada ao método RegisterChanges.

Declaração
```
procedure UnRegisterChanges(Value: TChangeLink);
```

Componentes aos quais se aplica:
TListImages

UnregisterEvents

Descrição

Este método cancela o registro dos eventos definidos pela propriedade Events do componente.

Declaração
```
procedure UnregisterEvents;
```

Componentes aos quais se aplica:
TIBEvents

Update

Descrição

No caso dos controles em geral, o método Update chama a função UpdateWindow da API do Windows para atualizar a sua exibição. No caso de objetos do tipo TFileListBox e TDirectoryListBox, o método força a atualização da lista de diretórios e arquivos. Para componentes dos tipos TFieldDefs e TIndexDefs, o método atualiza a propriedade Items.

Para objetos da classe TAction, executa o procedimento associado ao evento OnUpdate (se este estiver definido).

Declaração

Para objetos da classe TAction:
```
function Update: Boolean; override;
```

Para as demais classes e componentes:
```
procedure Update;
```

Exemplo

Para forçar um componente Memo1 do tipo TMemo a atualizar a sua exibição, use a seguinte linha de código:

```
Memo1.Update;
```

Componentes aos quais se aplica:

Todos os controles, objetos dos tipos TAction, TFieldDefs e TIndexDefs, e componentes dos tipos TDirectoryListBox e TFileListBox.

UpdateCursorPos

Descrição

Esse método ajusta a posição corrente do cursor no banco de dados de acordo com a posição atual do cursor no Borland Database Engine.

Declaração

```
procedure UpdateCursorPos;
```

Componentes aos quais se aplica:

```
TTable, TStoredProc e TQuery
```

UpdateRecord

Descrição

Esse método atualiza os registros em todos os componentes do tipo TDataSource.

Declaração

```
procedure UpdateRecord;
```

Componentes aos quais se aplica:

```
TADODataset, TADOTable, TADOQuery, TADOStoredProc, TIBDataset, TIBTable, TIBQuery,
TIBStoredProc, TTable, TQuery , TStoredProc, TClientDataset, TIBClientDataset, TSimpleDataset
e TTable
```

ValidateEdit

Descrição

Esse método analisa o valor da propriedade EditText para verificar a existência de espaços em branco nos quais é requerida a presença de um caractere. Se ocorrer, provoca uma exceção do tipo EDBEditError.

Declaração

```
procedure ValidateEdit;
```

Exemplo

O trecho de código a seguir usa o método ValidateEdit no evento OnExit de um componente DBEdit1 do tipo TDBEdit:

```
procedure TForm1.DBEdit1Exit(Sender: TObject);
begin
     ValidateEdit;
end;
```

Componentes aos quais se aplica:

```
TDBEdit e TMaskEdit
```

Write

Descrição

Esse método copia os bytes, armazenados na variável definida pelo parâmetro Buffer, no campo representado pelo objeto.

Declaração
```
function Write(const Buffer; Count: Longint): Longint; override;
```

Componentes aos quais se aplica:
TBlobStream

WriteBCDData

Descrição

Esse método escreve um valor do tipo Binário quando da ocorrência de um evento OnGetRow do componente de acesso vinculado ao relatório que está sendo gerado.

Declaração
```
function WriteBCDData(FormatData: String; NativeData: Currency): String;
```

Componentes aos quais se aplica:
TTRvCustomConnection, TRvDatasetConnection, TRvQueryConnection e TRvTableConnection.

WriteBlobData

Descrição

Esse método escreve um valor do tipo Blob (Binary Large Objects) quando da ocorrência de um evento OnGetRow do componente de acesso vinculado ao relatório que está sendo gerado.

Declaração
```
function WriteBlobData(var: Buffer; Len: Longint): String;
```

Componentes aos quais se aplica:
TTRvCustomConnection, TRvDatasetConnection, TRvQueryConnection e TRvTableConnection

WriteBool

Descrição

Esse método escreve um valor booleano (definido no parâmetro Ident) em uma seção (definida no parâmetro Section) de um arquivo INI.

Declaração
```
procedure WriteBool(const Section, Ident: string; Value: Boolean);
```

Componentes aos quais se aplica:
TIniFile

WriteBoolData

Descrição

Esse método escreve um valor do tipo Booleano quando da ocorrência de um evento OnGetRow do componente de acesso vinculado ao relatório que estã sendo gerado.

Declaração
```
function WriteBoolData(FormatData: String; NativeData: Boolean): String;
```

Componentes aos quais se aplica:
TTRvCustomConnection, TRvDatasetConnection, TRvQueryConnection e TRvTableConnection

WriteCurrData

Descrição

Esse método escreve um valor do tipo Currency quando da ocorrência de um evento OnGetRow do componente de acesso vinculado ao relatório que está sendo gerado.

Declaração
```
function WriteCurrData(FormatData: String; NativeData: Currency): String;
```

Componentes aos quais se aplica:
TTRvCustomConnection, TRvDatasetConnection, TRvQueryConnection e TRvTableConnection.

WriteDateTime

Descrição

Esse método escreve um valor do tipo DateTime quando da ocorrência de um evento OnGetRow do componente de acesso vinculado ao relatório que está sendo gerado.

Declaração
```
function WriteDateTime(FormatData: String; NativeData: TDateTime);
```

Componentes aos quais se aplica:
TTRvCustomConnection, TRvDatasetConnection, TRvQueryConnection e TRvTableConnection.

WriteFloatData

Descrição

Esse método escreve um valor do tipo float (real) quando da ocorrência de um evento OnGetRow do componente de acesso vinculado ao relatório que está sendo gerado.

Declaração
```
function WriteFloatData(FormatData: String; NativeData: Extended): String;
```

Componentes aos quais se aplica:
TTRvCustomConnection, TRvDatasetConnection, TRvQueryConnection e TRvTableConnection

WriteIntData

Descrição

Esse método escreve um valor do tipo inteiro quando da ocorrência de um evento OnGetRow do componente de acesso vinculado ao relatório que está sendo gerado.

Declaração
```
function WriteIntData(FormatData: String; NativeData: Integer): String;
```

Componentes aos quais se aplica:
TTRvCustomConnection, TRvDatasetConnection, TRvQueryConnection e TRvTableConnection

WriteInteger

Descrição

Esse método escreve um valor inteiro (definido no parâmetro Ident) em uma seção (definida no parâmetro Section) de um arquivo INI.

Declaração
```
procedure WriteInteger(const Section, Ident: string; Value: Longint);
```

Componentes aos quais se aplica:
TIniFile

WriteNullData

Descrição

Esse método escreve um valor do tipo NULL quando da ocorrência de um evento OnGetRow do componente de acesso vinculado ao relatório que está sendo gerado.

Declaração
```
function WriteNullData( no parameters );
```

Componentes aos quais se aplica:
TTRvCustomConnection, TRvDatasetConnection, TRvQueryConnection e TRvTableConnection

WriteStrData

Descrição

Esse método escreve um valor do tipo string quando da ocorrência de um evento OnGetRow do componente de acesso vinculado ao relatório que está sendo gerado.

Declaração
```
function WriteStrData(FormatData: String; NativeData: String): String;
```

Componentes aos quais se aplica:
TTRvCustomConnection, TRvDatasetConnection, TRvQueryConnection e TRvTableConnection

WriteString

Descrição

Esse método escreve o valor de uma string (definida no parâmetro Ident) em uma seção (definida no parâmetro Section) de um arquivo INI.

Declaração
```
procedure WriteString(const Section, Ident, Value: String);
```

Componentes aos quais se aplica:
TIniFile

ZoomToFit

Descrição

Esse método executa um zoom na pré-visualização do relatório, de forma a ocupar toda a área do componente.

Declaração
```
procedure ZoomToFit;
```

Componentes aos quais se aplica:
TQRPreview

ZoomToWidth

Descrição

Esse método executa um zoom na pré-visualização do relatório, de forma a ocupar toda a largura do componente.

Declaração
```
procedure ZoomToWidth;
```

Componentes aos quais se aplica:
TQRPreview.

Capítulo 43

Eventos

AfterCancel

Descrição

Esse evento ocorre após uma chamada ao método Cancel do componente.

Componentes aos quais se aplica:

```
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset,
TIBStoredproc, TClientDataset, TTable, TStoredProc TQuery, TDecisionQuery, TIBClientDataset e
TSQLClientDataset
```

AfterClose

Descrição

Esse evento ocorre após uma chamada ao método Close do componente.

Componentes aos quais se aplica:

```
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset,
TIBStoredproc, TClientDataset, TDecisionCube, TTable, TSQLTable, TSQLQuery, TSQLStoredProc,
TStoredProc, TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset
```

AfterConnect

Descrição

Esse evento ocorre após se estabelecer uma conexão a um banco de dados.

Componentes aos quais se aplica:

```
TDatabase, TIBDatabase, TADOConnection, TRDSConnection, TSQLConnection
```

AfterDelete

Descrição

Esse evento ocorre após uma chamada ao método Delete do componente.

Componentes aos quais se aplica:

```
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset,
TIBStoredproc, TClientDataset, TTable, TStoredProc TQuery, TDecisionQuery, TIBClientDataset e
TSQLClientDataset.
```

AfterDetail

Descrição

Esse evento ocorre logo após a impressão de um registro acessado através de um componente do tipo TQRBand.

Componentes aos quais se aplica:

```
TQuickReport
```

AfterDisconnect

Descrição

Esse evento ocorre imediatamente após se encerrar uma conexão a um banco de dados.

Componentes aos quais se aplica:

```
TDatabase, TIBDatabase, TADOConnection, TRDSConnection, TSQLConnection
```

AfterDispatch

Descrição

Esse evento ocorre quando o componente gera o conteúdo a ser enviado como mensagem de resposta a uma solicitação.

Componentes aos quais se aplica:
TXMLBroker e TWebDispatcher

AFTERDRAWVALUES

Descrição

Esse evento ocorre logo após os pontos de uma série terem sido plotados em um gráfico.

Componentes aos quais se aplica:
TChartSeries

AFTEREDIT

Descrição

Esse evento ocorre após uma chamada ao método Edit do componente.

Componentes aos quais se aplica:
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TTable, TStoredProc TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset.

AFTERINSERT

Descrição

Esse evento ocorre após uma chamada ao método Insert do componente.

Componentes aos quais se aplica:
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TTable, TStoredProc TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset.

AFTEROPEN

Descrição

Esse evento ocorre após uma chamada ao método Open do componente.

Componentes aos quais se aplica:
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TDecisionCube, TTable, TSQLTable, TSQLQuery, TSQLStoredProc, TStoredProc, TQuery, TDecisionQuery, , TIBClientDataset e TSQLClientDataset.

AFTERPOST

Descrição

Esse evento ocorre após uma chamada ao método Post do componente.

Componentes aos quais se aplica:
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TTable, TStoredProc TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset

AFTERPRINT

Descrição

Esse evento ocorre logo após a impressão em um componente do tipo TQRBand.

Componentes aos quais se aplica:
TQRBand

AfterRefresh

Descrição

Esse evento ocorre após uma chamada ao método Refresh do componente.

Componentes aos quais se aplica:

TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TDecisionCube, TTable, TSQLTable, TSQLQuery, TSQLStoredProc, TStoredProc, TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset

AfterScroll

Descrição

Esse evento ocorre imediatamente após se alterar o registro corrente, mediante uma chamada a um dos métodos de navegação do componente.

Componentes aos quais se aplica:

TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TDecisionCube, TTable, TSQLTable, TSQLQuery, TSQLStoredProc, TStoredProc, TQuery, TDecisionQuery, , TIBClientDataset e TSQLClientDataset.

BeforeCancel

Descrição

Esse evento ocorre no início da execução do método Cancel do componente.

Componentes aos quais se aplica:

TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TTable, TStoredProc TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset

BeforeClose

Descrição

Esse evento ocorre no início da execução do método Close do componente.

Componentes aos quais se aplica:

TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TDecisionCube, TTable, TSQLTable, TSQLQuery, TSQLStoredProc, TStoredProc, TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset

BeforeConnect

Descrição

Esse evento imediatamente antes de se estabelecer uma conexão a um banco de dados.

Componentes aos quais se aplica:

TDatabase, TIBDatabase, TADOConnection, TRDSConnection, TSQLConnection

BeforeDelete

Descrição

Esse evento ocorre no início da execução do método Delete do componente.

Componentes aos quais se aplica:

TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TTable, TStoredProc TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset.

BeforeDetail

Descrição

Esse evento ocorre imediatamente antes da impressão de um registro acessado através de um componente do tipo TQRBand.

Componentes aos quais se aplica:

```
TQuickReport
```

BeforeDisconnect

Descrição

Esse evento imediatamente antes de se encerrar uma conexão a um banco de dados.

Componentes aos quais se aplica:

```
TDatabase, TIBDatabase, TADOConnection, TRDSConnection, TSQLConnection
```

BeforeDispatch

Descrição

Esse evento ocorre quando o componente recebe uma mensagem de solicitação através do protocolo http.

Componentes aos quais se aplica:

```
TXMLBroker e TWebDispatcher
```

BeforeDrawValues

Descrição

Esse evento ocorre antes que os pontos de uma série sejam plotados em um gráfico.

Componentes aos quais se aplica:

```
TChartSeries
```

BeforeEdit

Descrição

Esse evento ocorre no início da execução do método Edit do componente.

Componentes aos quais se aplica:

```
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset,
TIBStoredproc, TClientDataset, TTable, TStoredProc TQuery, TDecisionQuery, TIBClientDataset e
TSQLClientDataset.
```

BeforeInsert

Descrição

Esse evento ocorre no início da execução do método Insert do componente.

Componentes aos quais se aplica:

```
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset,
TIBStoredproc, TClientDataset, TTable, TStoredProc TQuery, TDecisionQuery, TIBClientDataset e
TSQLClientDataset
```

BeforeOpen

Descrição

Esse evento ocorre no início da execução do método Open do componente.

Componentes aos quais se aplica:

TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TDecisionCube, TTable, TSQLTable, TSQLQuery, TSQLStoredProc, TStoredProc, TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset

BeforePost

Descrição

Esse evento ocorre no início da execução do método Post do componente.

Componentes aos quais se aplica:

TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TTable, TStoredProc TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset

BeforePrint

Descrição

Esse evento ocorre imediatamente antes da impressão em um componente do tipo TQRBand

Componentes aos quais se aplica:

TQRBand

BeforeRefresh

Descrição

Esse evento ocorre no início da execução do método Refresh do componente.

Componentes aos quais se aplica:

TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TDecisionCube, TTable, TSQLTable, TSQLQuery, TSQLStoredProc, TStoredProc, TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset

BeforeScroll

Descrição

Esse evento ocorre antes de se alterar o registro corrente, mediante uma chamada a um dos métodos de navegação do componente.

Componentes aos quais se aplica:

TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TDecisionCube, TTable, TSQLTable, TSQLQuery, TSQLStoredProc, TStoredProc, TQuery, TDecisionQuery, TIBClientDataset e TSQLClientDataset

DatabaseDisonnected

Descrição

Esse evento ocorre quando o banco de dados, acessado através do mecanismo Interbase Express, é desconectado da aplicação.

Componentes aos quais se aplica:

TIBDatabase

DatabaseDisonnecting

Descrição

Esse evento ocorre quando o banco de dados, acessado através do mecanismo Interbase Express, está sendo desconectado da aplicação.

Componentes aos quais se aplica:
`TIBDatabase`

DatabaseFree

Descrição

Esse evento ocorre quando o componente libera a memória por ele alocada.

Componentes aos quais se aplica:
`TIBDatabase`

OnActionExecute

Descrição

Este evento ocorre quando o método Execute de um objeto TAction é executado, e não é tratado pela lista de ações tratadas pelo componente.

Componentes aos quais se aplica:
`TApplicationEvents`

OnActionUpdate

Descrição

Este evento ocorre quando o método Update de um objeto TAction é executado, e não é tratado pela lista de ações tratadas pelo componente.

Componentes aos quais se aplica:
`TApplicationEvents`

OnActivate

Descrição

Para componentes do tipo TForm, o método OnActivate ocorre quando um formulário se torna ativo, isto é, quando ele recebe o foco.

Para componentes do tipo TApplication, o método OnActivate ocorre quando a aplicação se torna ativa, isto é, quando o Windows alterna entre outra aplicação e a representada pelo componente.

Exemplo

O trecho de código a seguir faz com que um formulário chamado Form1 do tipo TForm alterne a sua cor entre vermelho e amarelo cada vez que o formulário se torna ativo.

```
procedure TForm1.FormActivate(Sender: TObject);
begin
if Form1.Color = clRed then Form1.Color: = clYellow else Form1.Color: = clRed;
end;
```

Componentes aos quais se aplica:
`TApplication, TApplicationEvents e TForm`

OnActiveControlChange

Descrição

Para componentes do tipo TScreen, o método OnActiveControlChange ocorre quando muda o controle que tem o foco da aplicação (definido na propriedade ActiveControl).

Declaração

```
property OnActiveFormChange: TNotifyEvent;
```

Componentes aos quais se aplica:
`TScreen`

OnActiveFormChange

Descrição

Para componentes do tipo TScreen, o método OnActiveFormChange ocorre quando muda o formulário ativo da aplicação (definido na propriedade ActiveForm).

Declaração
```
property OnActiveFormChange: TNotifyEvent;
```

Componentes aos quais se aplica:
`TScreen`

OnAfterAdd

Descrição

Esse evento ocorre logo após um ponto ser adicionado a uma série.

Componentes aos quais se aplica:
`TChartSeries`

OnAfterClose

Descrição

Esse evento ocorre logo após o projeto de relatório representado pelo componente ser fechado.

Componentes aos quais se aplica:
`TRvProject`

OnAfterDraw

Descrição

Esse evento ocorre logo após o componente ter exibido todas as séries a ele associadas.

Componentes aos quais se aplica:
`TChart, TDBChart`

OnAfterOpen

Descrição

Esse evento ocorre logo após o projeto de relatório representado pelo componente ser aberto.

Componentes aos quais se aplica:
`TRvProject`

OnAfterPivot

Descrição

Esse evento ocorre logo após uma alteração no estado do pivô corrente.

Componentes aos quais se aplica:
`TDecisionSource`

OnAfterPrint

Descrição

Esse evento ocorre logo após ao impressão do relatório representado pelo componente.

Componentes aos quais se aplica:
TBaseReport, TRvNDRWriter,TRvRenderPreview e TRvRenderPrinter

OnAfterRender

Descrição

Esse evento ocorre logo após o formulário ser exibido como uma página HTML.

Componentes aos quais se aplica:
TIWForm

OnApply

Descrição

Para componentes do tipo TFontDialog, o método OnApply ocorre quando o usuário clica no botão Apply da caixa de diálogo.

Exemplo

O trecho de código a seguir faz com que a fonte selecionada em uma caixa de diálogo do tipo TFontDialog seja aplicada ao texto exibido num controle chamado Edit1 do tipo Tedit quando o usuário seleciona o botão Apply.

```
procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: Word);
begin
Edit1.Font: = FontDialog1.Font;
end;
```

Componentes aos quais se aplica:
TFontDialog

OnBeforeAdd

Descrição

Esse evento ocorre antes de um ponto ser adicionado a uma série.

Componentes aos quais se aplica:
TChartSeries

OnBeforeClose

Descrição

Esse evento ocorre imediatamente após o projeto de relatório representado pelo componente ser fechado.

Componentes aos quais se aplica:
TRvProject

OnBeforeOpen

Descrição

Esse evento ocorre imediatamente antes de o projeto de relatório representado pelo componente ser aberto.

Componentes aos quais se aplica:
TRvProject

OnBeforePivot

Descrição

Esse evento ocorre imediatamente antes de uma alteração no estado do pivô corrente.

Componentes aos quais se aplica:
`TDecisionSource`

OnBeforePrint

Descrição

Esse evento ocorre imediatamente antes da impressão do relatório representado pelo componente.

Componentes aos quais se aplica:
`TBaseReport, TRvNDRWriter,TRvRenderPreview e TRvRenderPrinter`

OnBeginRetrieval

Descrição

Esse evento ocorre quando um documento HTML começa a ser carregado.

Componentes aos quais se aplica:
`THTML`

OnBeginTransComplete

Descrição

Esse evento ocorre após o início de uma transação.

Componentes aos quais se aplica:
`TADOConnection`

OnCalcFields

Descrição

Esse evento ocorre no início da leitura de um registro de um banco de dados.

Componentes aos quais se aplica:
`TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery`

OnCancel

Descrição

Esse evento ocorre ao se cancelar a atualização de um registro.

Componentes aos quais se aplica:
`TIWDBNavigator`

OnChange

Descrição

O evento OnChange ocorre quando uma propriedade de um componente ou objeto é alterada durante a execução do aplicativo.

Exemplo

O trecho de código a seguir faz com que uma caixa de edição alterne sua cor entre vermelho e amarelo quando o usuário altera seu texto (sua propriedade Text) durante a execução do aplicativo.

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
     if Edit1.Color = clRed then Edit1.Color:= clYellow else Edit1.Color:= clRed;
end;
```

Componentes aos quais se aplica:

TActionList, TBitmap, TBrush, TCanvas, TComboBox, TFont, TGraphic, TGraphicsObject, TMetafile, TPen, TPicture, TStringList objects, TComboBox, TDirectoryListBox, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TIWCombobox, TIWDBCombobox, TIWDBLookupCombobox, TIWDBLookupListbox, TMaskEdit, TMemo, TQuery, TDecisionQuery, TScrollBar, TClientDataset, TTable, TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField, TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField, TStringField, TTimeField, TVarBytesField, TWordField e TTabSet, TDDEClientItem e TDDEServerItem

OnChanging

Descrição

Para componentes do tipo Tcanvas, esse evento ocorre imediatamente antes de se modificar o gráfico exibido por um objeto do tipo TCanvas.

Para componentes do tipo TUpDown, esse evento ocorre quando o valor armazenado na propriedade Position do controle está para ser alterado, após o usuário selecionar uma das setas do controle.

Para componentes do tipo TTreeView e TListView, esse evento ocorre quando se altera o item selecionado no controle.

Para componentes do tipo TTabControl e TPageControl, esse evento ocorre quando se altera a guia selecionada, e antes que esta receba o foco.

Componentes aos quais se aplica:

TCanvas, TUpDown, TListView, TTreeView, TTabControl e TPageControl

OnClearValues

Descrição

Esse evento ocorre logo após os pontos de uma série terem sido removidos.

Componentes aos quais se aplica:

TChartSeries

OnClick

Descrição

O evento OnClick ocorre quando o usuário dá um clique com o botão do mouse sobre um componente, isto é, quando o usuário pressiona e solta o botão primário do mouse (geralmente o botão esquerdo). Esse evento também ocorre quando:

- O usuário seleciona um item de uma Grade, Outline, caixa de lista ou caixa combo com uma das teclas de seta.
- O usuário pressiona a barra de espaço enquanto um botão ou caixa de verificação tem o foco da aplicação.
- O usuário pressiona a tecla Enter quando o formulário ativo tem um botão com a propriedade Default igual a True.
- O usuário pressiona a tecla Esc quando o formulário ativo tem um botão com a propriedade Cancel igual a True.
- O usuário pressiona a tecla aceleradora correspondente a um botão ou caixa de verificação.
- A propriedade Checked de um botão de rádio é igual a True.
- A propriedade Checked de uma caixa de verificação tem seu valor alterado.

- O método Click de um item de menu é executado.
- O usuário dá um clique sobre uma área em branco de um formulário.
- O usuário dá um clique sobre uma das setas em um controle do tipo TUpDown.
- O usuário dá um clique sobre um dos pontos de uma série representada por um objeto da classe TChartSeries.

Exemplo

O trecho de código a seguir faz com que um botão chamado Button1 exiba em sua propriedade Caption o número de vezes que o usuário clicou sobre ele.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     n:= n + 1;
     Button1.Caption:= IntToStr(n) + 'Cliques';
end;
```

Componentes aos quais se aplica:

```
TBitBtn, TButton, TChartSeries, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit,
TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDBNavigator, TDBRadioGroup,
TDBText, TDirectoryListBox, TDrawGrid, TDriveComboBox, TFileListBox, TFilterComboBox, TForm,
TFrame, TGroupBox, TImage, TIWButton, TIWCheckbox, TIWDBCheckbox, TIWImage, TIWImagefile,
TIWLink, TIWURL, TLabel, TListBox, TListView, TMaskEdit, TMediaPlayer, TMemo, TMenuItem,
TNotebook, TOutline, TPaintBox, TDecisionPivot,TPanel, TRadioButton, TScrollBar, TScrollBox,
TSpeedButton, TStatusBar, TStringGrid, TTabSet, TtreeView, THTML e TUpDown
```

OnClickAxis

Descrição

Esse evento ocorre quando o usuário clica sobre um dos eixos do gráfico exibido no componente.

Componentes aos quais se aplica:

```
TChart, TDBChart
```

OnClickLegend

Descrição

Esse evento ocorre quando o usuário clica sobre a área retangular que define a legenda do gráfico exibido no componente.

Componentes aos quais se aplica:

```
TChart, TDBChart
```

OnClose

Descrição

Para componentes do tipo TForm, esse evento define que manipulador de eventos deve ser chamado quando um formulário está prestes a ser fechado. O manipulador de eventos depende do parâmetro Action, que pode ter um dos seguintes valores:

- caNone: O formulário não pode ser fechado.
- caHide: O formulário não é fechado, mas não é mais visível.
- caFree: O formulário é fechado e toda memória alocada é liberada.
- caMinimize: O formulário não é fechado, porém é minimizado.

Para componentes dos tipos TDDEClientConv e TDDEServConv esse evento ocorre quando a conversação é encerrada.

Para componentes do tipo TAnimate, esse evento ocorre quando o componente é fechado.

Exemplo

O trecho de código a seguir faz com que um diálogo de confirmação seja exibido quando o usuário tenta fechar um formulário chamado Form1 do tipo TForm.

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
if MessageDlg('Fechar o formulário ?', mtConfirmation,[mbYes, mbNo], 0) = mrYes
then
Action: = caFree
else
Action: = caNone;
end;
```

Componentes aos quais se aplica:

TAnimate, TDDEClientConv, TDDEServConv e TForm

OnCloseQuery

Descrição

Para componentes do tipo TForm, esse evento é chamado quando se tenta fechar um formulário, e tem uma propriedade booleana chamada CanClose cujo valor define se o formulário pode ou não ser fechado.

Exemplo

O trecho de código a seguir faz com que um diálogo de confirmação seja exibido quando o usuário tenta fechar um formulário chamado Form1 do tipo TForm.

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
if MessageDlg('Fechar o formulário?', mtConfirmation,[mbOk, mbCancel], 0) = mrCancel then
CanClose: = False;
end;
```

Componentes aos quais se aplica:

TForm

OnColEnter

Descrição

Esse evento ocorre quando o usuário seleciona uma célula de uma determinada coluna, ou move-se para essa coluna com a tecla Tab.

Exemplo

O trecho de código a seguir faz com que um componente Label1 do tipo TLabel exiba uma mensagem quando o usuário se mover para uma determinada coluna.

```
procedure TForm1.DBGrid1ColEnter(Sender: TObject);
begin
      Label1.Caption:= 'A coluna foi selecionada';
end;
```

Componentes aos quais se aplica:

TDBCtrlGrid e TDBGrid

OnColExit

Descrição

Esse evento ocorre quando o usuário seleciona uma célula de uma outra coluna, ou move-se para outra coluna com a tecla Tab.

Exemplo

O trecho de código a seguir faz com que um componente Label1 do tipo TLabel exiba uma mensagem quando o usuário se mover para uma outra coluna:

```
procedure TForm1.DBGrid1ColExit(Sender: TObject);
begin
      Label1.Caption:= 'Outra coluna foi selecionada';
end;
```

Componentes aos quais se aplica:

`TDBCtrlGrid e TDBGrid`

OnCollapse

Descrição

O evento OnCollapse ocorre quando um item em um componente do tipo TOutline oculta os seus subitens.

Exemplo

O trecho de código a seguir exibe uma mensagem quando ocorre um evento OnCollapse.

```
procedure TForm1.Outline1Collapse(Sender: TObject; Index: Longint);
begin
ShowMessage('Ocultando os subitens de um item em um componente TOutline');
end;
```

Componentes aos quais se aplica:

`TOutline`

OnColumnClick

Descrição

Esse evento ocorre quando o usuário seleciona, com o botão esquerdo do mouse, o cabeçalho de uma coluna do componente.

Componentes aos quais se aplica:

`TListView`

OnColumnMoved

Descrição

O evento OnColumnMoved ocorre quando o usuário movimenta uma coluna com o mouse.

Exemplo

O trecho de código a seguir bloqueia movimentos posteriores de uma coluna.

```
procedure TForm1.StringGrid1ColumnMoved(Sender: TObject; FromIndex, ToIndex: Longint);
begin
StringGrid1.Options: = StringGrid1.Options - [goColMoving];
end;
```

Observação: Para que o evento ocorra é preciso que, inicialmente, a propriedade Options inclua a subpropriedade goColMoving.

Componentes aos quais se aplica:
`TDrawGrid e TStringGrid`

OnCommitTransComplete

Descrição

Esse evento ocorre após o término de uma transação.

Componentes aos quais se aplica:
`TADOConnection`

OnCompare

Descrição

Esse evento ocorre durante a reordenação dos itens do componente.

Componentes aos quais se aplica:
`TListView e TTreeView`

OnConnect

Descrição

Esse evento ocorre assim que se estabelece uma conexão ao servidor.

Componentes aos quais se aplica:
`TNMDayTime, TNMEcho, TNMFinger, TNMFTP, TNMHTTP, TNMMSG, TNMNNTP, TNMPop3, TNMSTRM, TPowerSock`

OnConnectComplete

Descrição

Esse evento ocorre quando se estabelece uma conexão a um banco de dados através do mecanismo Activex Data Objects.

Componentes aos quais se aplica:
`TADOConnection`

OnConnectionFailed

Descrição

Esse evento ocorre assim que uma tentativa de conexão falha.

Componentes aos quais se aplica:
`TNMDayTime, TNMEcho, TNMFinger, TNMFTP, TNMHTTP, TNMMSG, TNMNNTP, TNMPop3, TNMSTRM, TNMTimer, TPowerSock`

OnConnectionRequired

Descrição

Esse evento ocorre sempre que se executa um método que requer uma conexão ao servidor.

Componentes aos quais se aplica:
`TNMEcho, TNMPop3`

OnCreate

Descrição

O evento OnCreate especifica que manipuladores de eventos devem ser chamados quando o formulário é criado. É nesse evento que deve ser escrito o código que inicializa propriedades e realiza qualquer processamento antes que o usuário possa interagir com o formulário.

Quando um formulário está sendo criado e a sua propriedade Visible é igual a True, ocorrem os seguintes eventos, na ordem listada: OnActivate, OnShow, OnCreate e OnPaint.

Exemplo

O trecho de código a seguir faz com que um formulário chamado Form1 seja criado na cor Azul.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Form1.Color: = clBlue;
end;
```

Componentes aos quais se aplica:

TForm e TIWForm

OnDataChange

Descrição

O evento OnDataChange ocorre quando a propriedade State do componente deixa de ter o valor dsInactive, ou recebe uma notificação de um componente de exibição de dados durante a execução do aplicativo.

Componentes aos quais se aplica:

TDataSource

OnDblClick

Descrição

O evento OnDblClick ocorre quando o usuário dá um duplo clique com o botão do mouse sobre um componente, isto é, quando o usuário pressiona e solta duas vezes num curto intervalo de tempo o botão primário do mouse (geralmente o botão esquerdo).

Exemplo

O trecho de código a seguir faz com que uma mensagem seja exibida quando o usuário der um duplo clique sobre o formulário de uma aplicação.

```
procedure TForm1.FormClick(Sender: TObject);
begin
     MessageDlg('Você deu um duplo-clique sobre o formulário', mtInformation, [mbOk], 0);
end;
```

Componentes aos quais se aplica:

TComboBox, TDBComboBox, TDBEdit, TDBCtrlGrid, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDBNavigator, TDBText, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TForm, TFrame, TGroupBox, TImage, TLabel, TListBox, TMaskEdit, TMemo, TNotebook, TOLEContainer, TOutline, TPaintBox, TDecisionPivot,TPanel, TRadioButton, TScrollBox, TSpeedButton, THTML e TStringGrid

OnDeActivate

Descrição

Para componentes do tipo TApplication, o método OnDeActivate ocorre quando a aplicação perde o foco, isto é, quando o Windows alterna para outra aplicação diferente da representada pelo componente.

Exemplo

O trecho de código a seguir faz com que um formulário chamado Form1 do tipo TForm de uma aplicação seja minimizado quando o Windows seleciona outra aplicação como ativa.

```
procedure TForm1.AppDeactivate(Sender: TObject);
begin
Application.Minimize;
end;
```

Componentes aos quais se aplica:
`TApplication, TApplicationEvents`

OnDefaultAction

Descrição

Esse evento ocorre logo após a execução default do formulário exibido, como uma página HTML sendo executada.

Componentes aos quais se aplica:
`TIWForm`

OnDelete

Descrição

Esse evento ocorre ao se remover um registro.

Componentes aos quais se aplica:
`TIWDBNavigator`

OnDeletion

Descrição

Esse evento ocorre durante a eliminação de um item do componente.

Componentes aos quais se aplica:
`TListView e TTreeView`

OnDesignerSave

Descrição

Se este evento for definido, um item de menu intitulado “Save” será exibido no editor de relatórios, permitindo que o mesmo seja salvo pelo usuário final após realizar alterações no mesmo.

Componentes aos quais se aplica:
`TRvProject`

OnDesignerSaveAs

Descrição

Se este evento for definido, um item de menu intitulado “Save As” será exibido no editor de relatórios, permitindo que o mesmo seja salvo pelo usuário final em outro nome ou formato.

Componentes aos quais se aplica:
`TRvProject`

OnDesignerShow

Descrição

Esse evento ocorre imediatamente antes de o editor de relatórios ser aberto e exibido.

Componentes aos quais se aplica:
`TRvProject`

OnDestroy

Descrição

O evento OnDestroy ocorre quando um formulário está para ser destruído, como resposta a um método como Destroy, Free ou Release.

Exemplo

O trecho de código a seguir faz com que a memória alocada para um ponteiro seja liberada ao se fechar um formulário chamado Form1.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Dispose(pointer);
end;
```

Componentes aos quais se aplica:

`TForm e TIWForm`

OnDialectDowngradeWarning

Descrição

O evento OnDialectDowngradeWarning ocorre quando se altera a versão do dialeto SQL usado por uma aplicação cliente durante o acesso a um banco de dados através do mecanismo Interbase Express.

Componentes aos quais se aplica:

`TIBDatabase`

OnDisConnect

Descrição

Para objetos das classes TNMDayTime, TNMEcho, TNMFinger, TNMFTP, TNMHTTP, TNMMSG, TNMNNTP, TNMPop3, TNMSTRM, TpowerSock, esse evento ocorre assim que se encerra uma conexão ao servidor.

Para objetos da classe TADOConnection este evento ocorre ao se encerrar uma conexão a um banco de dados através do mecanismo Activex Data Objects.

Componentes aos quais se aplica:

`TADOConnection, TNMDayTime, TNMEcho, TNMFinger, TNMFTP, TNMHTTP, TNMMSG, TNMNNTP, TNMPop3, TNMSTRM, TPowerSock`

OnDockDrop

Descrição

O evento OnDockDrop ocorre quando o usuário solta um componente que está sendo ancorado sobre outro componente.

Componentes aos quais se aplica:

`TPanel, TChart, TDBChart, TDecisionGraph, TForm, TFrame, TDBCtrlGrid, TdrawGrid, TDecisionGrid e TStringGrid`

OnDockOver

Descrição

O evento OnDragOver ocorre quando o usuário arrasta um componente sobre outro numa operação de drag-e-dock.

Componentes aos quais se aplica:

`TPanel, TChart, TDBChart, TDecisionGraph, TForm, TFrame, TDBCtrlGrid, TdrawGrid, TDecisionGrid e TStringGrid`

OnDragDrop

Descrição

O evento OnDragDrop ocorre quando o usuário solta um componente que está sendo arrastado sobre outro componente. Observe que esse evento corresponde ao objeto receptor (Sender) e não ao que está sendo arrastado (Source).

Exemplo

Coloque em um formulário chamado Form1 três componentes do tipo TLabel chamados Label1, Label2 e Label3, com propriedade Color igual a clYellow, clRed e clBlue respectivamente, e a propriedade DragMode igual a dmAutomatic. Coloque um componente do tipo TListBox chamado ListBox1 e defina seu evento OnDragDrop da seguinte forma:

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
       ListBox1.Color:= (Source as TLabel).Color;
end;
```

Quando você arrastar e soltar os componentes TLabel sobre o ListBox, este assumirá a cor do componente que foi arrastado.

Componentes aos quais se aplica:

TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBCtrlGrid, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDBNavigator, TDBText, TDBRadioGroup, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TForm, TFrame, TGroupBox, TImage, TListBox, TMaskEdit, TMemo, TNotebook, TOLEContainer, TOutline, TPaintBox, TDecisionPivot,TPanel, TRadioButton, TScrollBar, TScrollBox, TShape, TStringGrid e TTabSet

OnDragOver

Descrição

O evento OnDragOver ocorre quando o usuário arrasta um componente sobre outro numa operação de drag-e-drop.

Exemplo

Se você não quiser que um componente TListBox permita que um componente qualquer seja arrastado sobre ele, basta incluir a seguinte linha de código em seu evento

```
OnDragOver:
Accept:= False;
```

Componentes aos quais se aplica:

TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBCtrlGrid, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDBNavigator, TDBText, TDBRadioGroup, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TForm, TFrame, TGroupBox, TImage, TListBox, TMaskEdit, TMemo, TNotebook, TOLEContainer, TOutline, TPaintBox, TDecisionPivot,TPanel, TRadioButton, TScrollBar, TScrollBox, TShape, TStringGrid e TTabSet

OnDrawCell

Descrição

O evento OnDrawItem ocorre quando uma célula do componente precisa ser redesenhada.

Exemplo

O trecho de código a seguir desenha um retângulo de foco em torno de cada uma das células de um componente StringGrid1 do tipo TStringGrid.

```
procedure TForm1.StringGrid1DrawCell(Sender: TObject; Col, Row: Longint;
Rect: TRect; State: TGridDrawState);
begin
StringGrid1.Canvas.DrawFocusRect(Rect);
end;
```

Componentes aos quais se aplica:

TDrawGrid e TStringGrid

OnDrawItem

Descrição

O evento OnDrawItem ocorre quando um item de um componente owner-draw precisa ser redesenhado.

Exemplo

O trecho de código abaixo define o evento OnDrawItem para um componente chamado ListBox1 do tipo TListBox com propriedade Style igual a lbOwnerDrawFixed e desenha um bitmap à esquerda de cada string.

```
procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect: TRect;
State: TOwnerDrawState);
var
Bitmap: TBitmap;
Offset: Integer;
begin
with (Control as TListBox).Canvas do
begin
FillRect(Rect);
Offset: = 2;
Bitmap: = TBitmap(Items.Objects[Index]);
if Bitmap <> nil then
begin
BrushCopy(Bounds(Rect.Left + 2, Rect.Top, Bitmap.Width, Bitmap.Height), Bitmap,
Bounds(0, 0, Bitmap.Width, Bitmap.Height), clRed);
Offset: = Bitmap.width + 6;
end;
TextOut(Rect.Left + Offset, Rect.Top, Items[Index])
end;
end;
```

Componentes aos quais se aplica:

TComboBox, TDBComboBox, TDBListBox, TListBox e TOutline

OnDrawTab

Descrição

O evento OnDrawTab ocorre quando uma guia do controle precisa ser redesenhada, e o controle possui a propriedade Style igual a tsOwnerDraw.

Exemplo

O trecho de código a seguir exibe uma mensagem quando ocorre o evento OnDrawTab.

```
procedure TForm1.TabSet1DrawTab(Sender: TObject; TabCanvas: TCanvas; R: TRect;
Index: Integer; Selected: Boolean);
begin
ShowMessage('A guia vai ser redesenhada');
end;
```

Componentes aos quais se aplica:

TTabSet

OnDropDown

Descrição

O evento OnDropDown ocorre quando o usuário exibe o conteúdo de uma caixa de listagem ou caixa combo.

Exemplo

O trecho de código a seguir faz com que a caixa de listagem de um componente do tipo TComboBox alterne sua cor entre vermelho e amarelo sempre que for exibida.

```
procedure TForm1.ComboBox1DropDown(Sender: TObject);
begin
if ComboBox1.Color = clRed then ComboBox1.Color: = clYellow else ComboBox1.Color: = clRed;
end;
```

Componentes aos quais se aplica:

TComboBox, TDBListBox, TDBComboBox, TDBLookupCombo e TListBox

OnEdit

Descrição

Esse evento ocorre ao se iniciar a edição de um registro.

Componentes aos quais se aplica:

TIWDBNavigator

OnEditError

Descrição

O evento OnEditError ocorre quando uma exceção é gerada ao se tentar editar um registro.

Componentes aos quais se aplica:

TClientDataset, TTable, TQuery, TDecisionQuery

OnEndDock

Descrição

O evento OnEndDock ocorre quando o usuário termina de arrastar um componente sobre outro, sendo ancorado em outro objeto. Esse evento corresponde ao componente que está sendo arrastado.

Componentes aos quais se aplica:

TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBCtrlGrid, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDBNavigator, TDBText, TDBRadioGroup, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TFrame, TGroupBox, TImage, TListBox, TMaskEdit, TMemo, TNotebook, TOLEContainer, TOutline, TDecisionPivot,TPanel, TRadioButton, TScrollBar, TScrollBox, TShape, TStringGrid e TTabSet

OnEndDrag

Descrição

O evento OnEndDrag ocorre quando o usuário termina de arrastar um componente sobre outro. Esse evento corresponde ao componente que está sendo arrastado.

Exemplo

Se você quiser que um componente TLabel chamado Label1 se movimente 20 pixels para a direita ao terminar de ser arrastado, basta incluir a seguinte linha de código em seu evento OnEndDrag:

```
Label1.Left:= Label1.Left + 10;
```

Componentes aos quais se aplica:

```
TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBCtrlGrid, TDBEdit,
TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDBNavigator, TDBText, TDBRadioGroup,
TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TFrame,
TGroupBox, TImage, TListBox, TMaskEdit, TMemo, TNotebook, TOLEContainer, TOutline,
TDecisionPivot,TPanel, TRadioButton, TScrollBar, TScrollBox, TShape, TStringGrid e TTabSet
```

OnEndOfRecordset

Descrição

Esse evento ocorre quando o cursor atinge o último registro de um Recordset (conjunto de registros de uma tabela ou resultantes de uma consulta SQL).

Componentes aos quais se aplica:

```
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset,
TIBStoredproc, TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery
```

OnEndPage

Descrição

Esse evento ocorre imediatamente antes do surgimento de uma quebra de página no relatório.

Componentes aos quais se aplica:

```
TQuickReport
```

OnEnter

Descrição

O evento OnEnter ocorre quando um componente se torna ativo, isto é, assim que ele recebe o foco da aplicação.

Exemplo

Se você incluir a linha de código abaixo no evento OnEnter de um componente Edit1 do tipo TEdit, ele exibirá o texto 'Estou ativo' sempre que receber o foco da aplicação:

```
Edit1.text:= 'Estou ativo';
```

Componentes aos quais se aplica:

```
TBitBtn, TButton, TChart, TDBChart,TCheckBox, TComboBox, TDBCheckBox, TDBComboBox,
TDBCtrlGrid, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo,
TDBNavigator, TDBRadioGroup, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit,
TFileListBox, TFilterComboBox, TForm, TFrame, TGroupBox, TListBox, TMaskEdit, TMediaPlayer,
TMemo, TNotebook, TOLEContainer, TOutline, TRadioButton, TRadioGroup, TScrollBar, TScrollBox,
TStringGrid, TTabbedNotebook, TTabSet
```

OnEndRetrieval

Descrição

Esse evento ocorre quando um documento HTML termina de ser carregado.

Componentes aos quais se aplica:

```
THTML
```

OnEventAlert

Descrição

Esse evento ocorre quando se verifica a ocorrência de um evento do próprio banco de dados do Interbase, acessado através do mecanismo Interbase Express, a ser tratado pela aplicação.

Componentes aos quais se aplica:
TIBEvents

OnException

Descrição

O evento OnException é ativado quando ocorre uma exceção não manipulada pela aplicação.

Componentes aos quais se aplica:
TApplication, TApplicationEvents

OnExecute

Descrição

O evento OnExecute ocorre quando o procedimento associado ao evento a ele vinculado é executado.

Componentes aos quais se aplica:
TAction, TActionList

OnExecuteComplete

Descrição

O evento OnExecuteComplete ocorre quando um comando é executado na manipulação de registros de um banco de dados acessado através do mecanismo Activex Data Objects.

Componentes aos quais se aplica:
TADOConnection

OnExecuteMacro

Descrição

O evento OnExecuteMacro ocorre quando a aplicação-cliente envia uma macro para ser executada na aplicação servidora em uma conversação DDE.

Componentes aos quais se aplica:
TDDEServerConv

OnExit

Descrição

O evento OnExit ocorre quando um componente perde o foco da aplicação.

Exemplo

Se você incluir a linha de código abaixo no evento OnExit de um componente Edit1 do tipo TEdit, ele não exibirá nenhum texto quando perder o foco da aplicação:

```
Edit1.text:= '';
```

Componentes aos quais se aplica:
TBitBtn, TButton, TChart, TDBChart, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBCtrlGrid, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDBNavigator, TDBRadioGroup, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TForm, TFrame, TGroupBox, TListBox, TMaskEdit, TMediaPlayer, TMemo, TNotebook, TOLEContainer, TOutline, TRadioButton, TRadioGroup, TScrollBar, TScrollBox, TStringGrid, TTabbedNotebook, TTabSet

OnExpand

Descrição

O evento OnExpand ocorre quando um item em um componente do tipo TOutline exibe os seus subitens.

Exemplo

O trecho de código a seguir exibe uma mensagem quando ocorre um evento OnExpand.

```
procedure TForm1.Outline1Expand(Sender: TObject; Index: Longint);
begin
ShowMessage('Exibindo os subitens de um item em um componente TOutline');
end;
```

Componentes aos quais se aplica:

`TOutline`

ONFETCHCOMPLETE

Descrição

Esse evento ocorre quando se carrega todos os registros de um Recordset (conjunto de registrios de uma tabela ou resultantes de uma consulta SQL)

Componentes aos quais se aplica:

`TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery`

ONFETCHPROGRESS

Descrição

Esse evento ocorre enquanto se carrega todos os registros de um Recordset (conjunto de registrios de uma tabela ou resultantes de uma consulta SQL)

Componentes aos quais se aplica:

`TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery`

ONFILTER

Descrição

Esse evento pode ser usado para fazer uma filtragem local de registros durante a preparação de um relatório.

Componentes aos quais se aplica:

`TQRDetailLink`

ONFILTERRECORD

Descrição

O evento OnFilterRecord ocorre quando se altera o registro corrente de uma tabela na qual está definido um filtro.

Componentes aos quais se aplica:

`TClientDataset, TTable, TQuery, TDecisionQuery`

ONFIND

Descrição

O evento OnFind ocorre quando o usuário seleciona o botão Find Next (Localizar próxima) em uma caixa de diálogo.

Componentes aos quais se aplica:

`TFindDialog e TReplaceDialog`

OnFirst

Descrição

Esse evento ocorre ao se deslocar o cursor do componente que indica o registro corrente para o primeiro registro.

Componentes aos quais se aplica:

TIWDBNavigator

OnGetEditMask

Descrição

O evento OnGetEditMask ocorre quando o texto de uma célula precisa ser exibido ou atualizado usando-se uma máscara.

Exemplo

O trecho de código a seguir faz com que as células de um componente StringGrid1 do tipo TStringGrid exibam texto com uma máscara para números de telefone:

```
procedure TForm1.StringGrid1GetEditMask(Sender: TObject; ACol,
ARow: Longint; var Value: OpenString);
begin
Value: = '!\(999\)000-0000;1';
end;
```

Componentes aos quais se aplica:

TDrawGrid e TStringGrid

OnGetEditText

Descrição

O evento OnGetEditText ocorre quando o texto de uma célula precisa ser exibido ou atualizado.

Exemplo

O trecho de código a seguir faz com que as células de um componente StringGrid1 do tipo TStringGrid exibam texto com indicação dos números da linha e coluna em que se localiza.

```
procedure TForm1.StringGrid1GetEditMask(Sender: TObject; ACol,
ARow: Longint; var Value: OpenString);
begin
Value: = 'Linha '+IntToStr(StringGrid1.Row)+' e Coluna '+IntToStr(StringGrid1.Col);
end;
```

Componentes aos quais se aplica:

TDrawGrid e TStringGrid

OnGetErrorResponse

Descrição

Esse evento ocorre quando são detectados erros ao se aplicar atualizações na aplicação servidora.

Componentes aos quais se aplica:

TXMLBroker

OnGetResponse

Descrição

Esse evento ocorre quando não são detectados erros ao se aplicar atualizações na aplicação servidora.

Componentes aos quais se aplica:
`TXMLBroker`

OnGetText

Descrição

O evento OnGetText é disparado quando se faz uma referência às propriedades DisplayText e Text.

Componentes aos quais se aplica:
`TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField, TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField, TStringField, TTimeField, TVarBytesField e TWordField`

OnHelp

Descrição

O evento OnHelp ocorre quando a aplicação vai exibir uma mensagem ou tela de auxílio.

Componentes aos quais se aplica:
`Tapplication, TApplicationEvents`

OnHide

Descrição

Esse evento ocorre imediatamente antes de um formulário se tornar invisível.

Exemplo

O trecho de código a seguir emite uma mensagem informando que um formulário vai se tornar invisível.

```
procedure TForm1.FormHide(Sender: TObject);
begin
ShowMessage('O formulário vai ficar invisível');
end;
```

Componentes aos quais se aplica:
`TForm`

OnHint

Descrição

O evento OnHint ocorre imediatamente antes de um controle exibir uma string de auxílio (definida pela sua propriedade Hint). Para objetos da classe TAction, sobrepõe o comportamento default dos controles e itens de menu associados.

Componentes aos quais se aplica:
`TAction, Tapplication, TApplicationEvents`

OnHTMLTag

Descrição

Esse evento ocorre quando é necessário substituir uma tag transparente por um valor durante a geração de uma página HTML a partir de um template de documento HTML.

Componentes aos quais se aplica:
`TIWApplet, TIWButton, TIWCheckBox, TIWComboBox, TIWControl, TIWDBCheckBox, TIWDBCombobox, TIWEdit, TIWDBFile, TIWDBGrid, TIWDBImage, TIWDBListbox, TIWDBLookupCombobox, TIWDBLookupListbox, TIWDBMemo, TIWDBNavigator, TIWDBText, TIWEdit, TIWFile, TIWGrid, TIWImage, TIWImageFile, TIWLabel, TIWLink, TIWList, TIWListbox, TIWMemo, TIWRectangle, TIWTimer, TIWTreeview, TIWURL, TMidasPageProducer e TPageproducer`

OnIdle

Descrição

O evento OnIdle ocorre quando a sua aplicação está executando código ou aguardando uma ação do usuário.

Componentes aos quais se aplica:

```
Tapplication, TApplicationEvents
```

OnIdleTimer

Descrição

O evento OnIdleTimer ocorre quando se desconecta o acesso a um banco de dados através do mecanismo Interbase Express quando o tempo ocioso de conexão ultrapassa o valor especificado na propriedade IdleTimer do componente responsável pelo acesso.

Componentes aos quais se aplica:

```
TIBDatabase, TIBTransaction
```

OnInfoMessage

Descrição

O evento OnInfoMessage ocorre quando alguma mensagem do servidor de dados é recebida através da conexão a um banco de dados acessado através do mecanismo Activex Data Objects.

Componentes aos quais se aplica:

```
TADOConnection
```

OnInsert

Descrição

Esse evento ocorre ao se inserir um novo registro.

Componentes aos quais se aplica:

```
TIWDBNavigator
```

OnKeyDown

Descrição

O evento OnKeyDown ocorre quando o usuário pressiona uma tecla enquanto o controle tem o foco da aplicação.

Exemplo

Se você incluir a linha de código abaixo no evento OnKeyDown de um componente Edit1 do tipo TEdit, ele exibirá o texto 'Uma tecla foi pressionada' após o caractere digitado quando o usuário pressionar qualquer tecla enquanto o componente possuir o foco da aplicação:

```
Edit1.text:= 'Uma tecla foi pressionada';
```

Componentes aos quais se aplica:

```
TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBCtrlGrid, TDBEdit,
TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDirectoryListBox, TDrawGrid,
TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TForm, TListBox, TMaskEdit, TMemo,
TOLEContainer, TOutline, THTML, TRadioButton, TScrollBar e TStringGridDeclaration
```

OnKeyPress

Descrição

O evento OnKeyPress ocorre quando o usuário pressiona uma tecla que corresponda a um caractere ASCII enquanto o controle tem o foco da aplicação.

Exemplo

Se você incluir a linha de código abaixo no evento OnKeyPress de um componente Edit1 do tipo TEdit, ele exibirá o texto 'Uma tecla ASCII foi pressionada' após o caractere digitado quando o usuário pressionar qualquer tecla que corresponda a um caractere ASCII enquanto o componente possuir o foco da aplicação:

```
Edit1.text:= 'Uma tecla ASCII foi pressionada';
```

Componentes aos quais se aplica:

```
TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBCtrlGrid, TDBEdit,
TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDirectoryListBox, TDrawGrid,
TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TForm, TListBox, TMaskEdit, TMemo,
TOLEContainer, THTML, TOutline, TRadioButton, TScrollBar e TStringGrid
```

OnKeyUp

Descrição

O evento OnKeyUp ocorre quando o usuário solta uma tecla enquanto o controle tem o foco da aplicação.

Exemplo

Se você incluir a linha de código abaixo no evento OnKeyUp de um componente Edit1 do tipo TEdit, ele exibirá o texto 'Uma tecla foi solta' quando o usuário liberar uma tecla enquanto o componente tiver o foco da aplicação:

```
Edit1.text:= 'Uma tecla foi solta';
```

Componentes aos quais se aplica:

```
TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage,
TDBListBox, TDBLookupCombo, TDBMemo, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit,
TFileListBox, TFilterComboBox, TForm, TListBox, TListView, TMaskEdit, TMemo, TOutline, THTML,
TRadioButton, TRichEdit, TScrollBar, TStringGrid, TTrackBar e TTreeView
```

OnLast

Descrição

Esse evento ocorre ao se deslocar o cursor do componente que indica o registro corrente para o último registro.

Componentes aos quais se aplica:

```
TIWDBNavigator
```

OnLogin

Descrição

O evento OnLogin ocorre quando se estabelece uma conexão a um banco de dados.

Componentes aos quais se aplica:

```
TDatabase, TADOConnection, TRDSConnection, TIBDatabase, TSQLConnection
```

OnLowCapacity

Descrição

Esse evento ocorre quando a memória alocada para o cache do componente (definida pela sua propriedade Capacity) não é suficiente.

Componentes aos quais se aplica:

TDecisionCube

OnMeasureItem

Descrição

O evento OnMeasureItem ocorre quando a aplicação precisa redesenhar um item em um componente. Para um componente do tipo TListBox, esse evento ocorre se ele possuir a propriedade Style igual a lbOwnerDrawVariable, e para um componente do tipo TComboBox, esse evento ocorre se ele possuir a propriedade Style igual a csOwnerDraw.

Esse evento passa três parâmetros que descrevem o item a ser medido:

- O controle que contém o item.
- O índice do item no controle.
- A altura do item (em pixels).

Exemplo

O trecho de código a seguir define o evento OnMeasureItem para um componente do tipo TListBox com propriedade Style igual a lbOwnerDrawn, cujos itens a serem exibidos são Bitmaps. Se a altura do Bitmap a ser exibido for maior que a altura default, ele define a altura do item como sendo igual à do Bitmap.

```
procedure TForm1.ListBox1MeasureItem(Control: TWinControl; Index: Integer;
var Height: Integer);
var
Bitmap: TBitmap;
begin
with Control as TListBox do
begin
Bitmap: = TBitmap(Items.Objects[Index]);
if Bitmap <> nil then
if Bitmap.Height > Height then Height: = Bitmap.Height;
end;
end;
```

Componentes aos quais se aplica:

TComboBox, TDBComboBox, TDBListBox e TListBox

OnMeasureTab

Descrição

O evento OnMeasureTab ocorre imediatamente antes do evento OnDrawTab quando a aplicação precisa redesenhar uma guia em um controle. Para um componente do tipo TTabSet, esse evento ocorre se ele tiver a propriedade Style igual a tsOwnerDraw.

Exemplo

O trecho de código a seguir exibe uma mensagem quando ocorre o evento

```
OnMeasureTab.
procedure TForm1.TabSet1DrawTab(Sender: TObject; TabCanvas: TCanvas; R:
TRect;
Index: Integer; Selected: Boolean);
begin
ShowMessage('A guia vai ser redesenhada');
end;
```

Componentes aos quais se aplica:

TTabSet

OnMessage

Descrição

O evento OnMessage ocorre quando a aplicação recebe uma mensagem do Windows.

Componentes aos quais se aplica:

`Tapplication, TApplicationEvents`

OnMinimize

Descrição

O evento OnMinimize ocorre quando a aplicação é minimizada.

Componentes aos quais se aplica:

`TApplicationEvents`

OnMouseDown

Descrição

O evento OnMouseDown ocorre quando o usuário pressiona o botão do mouse sobre um controle.

Exemplo

Se você quiser que um componente Label1 do tipo TLabel exiba as coordenadas do ponto em que o usuário pressionou um botão do mouse sobre um formulário chamado Form1, defina o evento OnMouseDown de Form1 da seguinte forma:

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
     Label1.Caption:= 'Botão do Mouse pressionado em X = '+ IntToStr(X) + ', Y = '+
IntToStr(Y);
end;
```

Componentes aos quais se aplica:

`TBitBtn, TButton, TChart, TDBChart, TCheckBox, TDBCheckBox, TDBEdit, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDBNavigator, TDBText, TDirectoryListBox, TDrawGrid, TEdit, TFileListBox, TForm, TIWImage, TIWImageFile`

OnMouseMove

Descrição

O evento OnMouseMove ocorre quando o usuário movimenta o mouse sobre um controle.

Exemplo

Se você quiser que um componente Label1 do tipo TLabel exiba as coordenadas do ponto em que o mouse se encontra sobre um formulário chamado Form1, defina o evento OnMouseMove de Form1 da seguinte forma:

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
Y: Integer);
begin
    Label1.Caption:= 'Mouse em X = '+ IntToStr(X) + ', Y = '+ IntToStr(Y);
end;
```

Componentes aos quais se aplica:

`TBitBtn, TButton, TChart, TDBChart, TCheckBox, TDBCheckBox, TDBEdit, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDBNavigator, TDirectoryListBox, TDrawGrid, TEdit, TFileListBox, TForm, TFrame, TGroupBox, THeaderControl, TImage, TLabel, TListBox, TListView, TMaskEdit, TMemo, TNotebook, TOutline, TPageControl, TPaintBox, TDecisionPivot,TPanel, TProgressBar, TRadioButton, TRichEdit, TTScrollBox, TShape, TSpeedButton, TStatusBar, TStringGrid, THTML, TTabControl, TTabSet, TTreeView e TUpDown`

OnMouseUp

Descrição

O evento OnMouseUp ocorre quando o usuário libera o botão do mouse sobre um controle.

Exemplo

Se você quiser que um componente Label1 do tipo TLabel exiba as coordenadas do ponto em que o usuário liberou um botão do mouse sobre um formulário chamado Form1, defina o evento OnMouseUp de Form1 da seguinte forma:

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
     Label1.Caption:= ' Botão do Mouse liberado em X = '+ IntToStr(X) + ', Y = '+ IntToStr(Y);
end;
```

Componentes aos quais se aplica:

TBitBtn, TButton, TChart, TDBChart., TCheckBox, TDBCheckBox, TDBEdit, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDBNavigator, TDBText, TDirectoryListBox, TDrawGrid, TEdit, TFileListBox, TForm, TFrame, TGroupBox, THTML, THeaderControl, TImage, TLabel, TListBox, TListView, TMaskEdit, TMemo, TNotebook, TOutline, TPageControl, TPaintBox, TDecisionPivot,TPanel, TRadioButton, TRichEdit, TScrollBox, TShape, TSpeedButton, TStatusBar, TStringGrid, TTabControl, TTabSet, TTreeView e TUpDown

OnMouseWheel

Descrição

Este evento ocorre quando o usuário seleciona o botão de rolagem de página do mouse.

Componentes aos quais se aplica:

TBitBtn, TButton, TChart, TDBChart, TCheckBox, TDBCheckBox, TDBEdit, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDBNavigator, TDBText, TDirectoryListBox, TDrawGrid, TEdit, TFileListBox, TForm, TFrame, TGroupBox, THTML, THeaderControl, TImage, TLabel, TListBox, TListView, TMaskEdit, TMemo, TNotebook, TOutline, TPageControl, TPaintBox, TDecisionPivot,TPanel, TRadioButton, TRichEdit, TScrollBox, TShape, TSpeedButton, TStatusBar, TStringGrid, TTabControl, TTabSet, TTreeView e TUpDown

OnMouseWheelDown

Descrição

Este evento ocorre quando o usuário seleciona o botão de rolagem de página do mouse de forma a rolar a página para baixo.

Componentes aos quais se aplica:

TBitBtn, TButton, TChart, TDBChart, TCheckBox, TDBCheckBox, TDBEdit, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDBNavigator, TDBText, TDirectoryListBox, TDrawGrid, TEdit, TFileListBox, TForm, TFrame, TGroupBox, THTML, THeaderControl, TImage, TLabel, TListBox, TListView, TMaskEdit, TMemo, TNotebook, TOutline, TPageControl, TPaintBox, TDecisionPivot,TPanel, TRadioButton, TRichEdit, TScrollBox, TShape, TSpeedButton, TStatusBar, TStringGrid, TTabControl, TTabSet, TTreeView e TUpDown

OnMouseWheelUp

Descrição

Este evento ocorre quando o usuário seleciona o botão de rolagem de página do mouse de forma a rolar a página para cima.

Componentes aos quais se aplica:

TBitBtn, TButton, TChart, TDBChart, TCheckBox, TDBCheckBox, TDBEdit, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDBNavigator, TDBText, TDirectoryListBox, TDrawGrid, TEdit, TFileListBox, TForm, TFrame, TGroupBox, THTML, THeaderControl, TImage, TLabel, TListBox, TListView, TMaskEdit, TMemo, TNotebook, TOutline, TPageControl, TPaintBox, TDecisionPivot,TPanel, TRadioButton, TRichEdit, TScrollBox, TShape, TSpeedButton, TStatusBar, TStringGrid, TTabControl, TTabSet, TTreeView e TUpDown

OnMoveComplete

Descrição

Esse evento ocorre quando se altera o registro corrente de um Recordset (conjunto de registros de uma tabela ou resultantes de uma consulta SQL).

Componentes aos quais se aplica:

TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery

OnNeedData

Descrição

Para componentes do tipo TQRDetailLink, esse evento pode ser usado para obter dados a serem impressos a partir de outra fonte que não seja um componente do tipo TDataSource.

Para componentes do tipo TQRGroup, esse evento pode ser usado para definir a quebra de grupos de componentes.

Componentes aos quais se aplica:

TQRDetailLink e TQRGroup

OnNewDimensions

Descrição

Esse evento ocorre quando se alteram os dados acessados pelo componente DecisionCube associado.

Componentes aos quais se aplica:

TDecisionSource

OnNewPage

Descrição

Esse evento ocorre sempre que uma nova página é gerada para o relatório representado pelo componente.

Componentes aos quais se aplica:

TBaseReport, TRvNDRWriter,TRvRenderPreview e TRvRenderPrinter.

OnNewRecord

Descrição

Esse evento ocorre quando um registro é adicionado ao banco de dados.

Componentes aos quais se aplica:

TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery

OnNext

Descrição

Esse evento ocorre ao se deslocar o cursor do componente que indica o registro corrente para o próximo registro.

Componentes aos quais se aplica:

```
TIWDBNavigator
```

OnNotify

Descrição

O evento OnNotify ocorre durante a finalização de um método em um controle do tipo TMediaPlayer, fazendo com que sua propriedade Notify seja igual a True.

Exemplo

O trecho de código a seguir exibe uma mensagem quando o evento OnNotify é executado.

```
procedure TForm1.MediaPlayer1Notify(Sender: TObject);
begin
ShowMessage('Ocorreu o evento OnNotify');
end;
```

Componentes aos quais se aplica:

```
TMediaPlayer
```

OnOpen

Descrição

Para componentes dos tipos TDDEClientConv e TDDEServerConv, esse evento ocorre quando uma conversação DDE é iniciada.

Para componentes do tipo TAnimate, esse evento ocorre quando o componente é aberto.

Componentes aos quais se aplica:

```
TAnimate, TDDEClientConv e TDDEServerConv
```

OnPaint

Descrição

O evento OnPaint ocorre quando um componente do tipo TForm ou TPaintBox precisa redesenhar o seu conteúdo.

Exemplo

O trecho de código a seguir faz com que um formulário se torne amarelo ao ser restaurado após ser minimizado ou ter sido tornado invisível.

```
procedure TForm1.FormPaint(Sender: TObject);
begin
Form1.Color: = clYellow;
end;
```

Componentes aos quais se aplica:

```
TForm e TPaintBox
```

OnPassword

Descrição

O evento OnPassword ocorre quando uma tabela do Paradox é aberta e o Borland Database Engine verifica a necessidade de se fornecer uma senha (Password).

Componentes aos quais se aplica:

```
TSession
```

OnPopup

Descrição

O evento OnPopup ocorre quando um menu flutuante é exibido.

Componentes aos quais se aplica:

`TPopup`

OnPokeData

Descrição

O evento OnPokeData ocorre quando a aplicação cliente transfere dados para a aplicação servidora em uma conversação DDE.

Componentes aos quais se aplica:

`TDDEServerItem`

OnPost

Descrição

Esse evento ocorre ao se gravar um registro.

Componentes aos quais se aplica:

`TIWDBNavigator`

OnPostClick

Descrição

O evento OnPostClick ocorre após a execução do código do evento OnClick.

Exemplo

O trecho de código a seguir exibe uma mensagem quando o evento OnPostClick é executado.

```
procedure TForm1.MediaPlayer1PostClick(Sender: Tobject; Button: TMPBtnType);
begin
ShowMessage('Evento OnPostClick');
end;
```

Componentes aos quais se aplica:

`TMediaPlayer`

OnPostError

Descrição

O evento OnPostError ocorre quando uma exceção é gerada ao se tentar gravar ou alterar um registro.

Componentes aos quais se aplica:

`TClientDataset, TTable, TQuery, TDecisionQuery`

OnPreview

Descrição

Esse evento ocorre durante a pré-visualização de um relatório.

Componentes aos quais se aplica:

`TQRPrinter`

OnPrint

Descrição

Para componentes dos tipos TQRLabel, TQRDBCalc e TQRDBText esse evento ocorre imediatamente antes da impressão do relatório representado pelo componente.

Para componentes dos tipos TBaseReport, TRvNDRWriter,TRvRenderPreview e TRvRenderPrinter esse evento ocorre imediatamente antes da impressão do corpo do relatório representado pelo componente.

Componentes aos quais se aplica:

```
TQRLabel, TQRDBCalc, TQRDBText, TBaseReport, TRvNDRWriter,TRvRenderPreview e TRvRenderPrinter
```

OnPrintFooter

Descrição

Esse evento ocorre imediatamente após a impressão do corpo do relatório representado pelo componente, e imediatamente antes da impressão do seu rodapé.

Componentes aos quais se aplica:

```
TBaseReport, TRvNDRWriter,TRvRenderPreview e TRvRenderPrinter
```

OnPrintHeader

Descrição

Esse evento ocorre imediatamente após a impressão do cabeçalho e imediatamente antes da impressão do corpo do relatório representado pelo componente.

Componentes aos quais se aplica:

```
TBaseReport, TRvNDRWriter,TRvRenderPreview e TRvRenderPrinter.
```

OnPrior

Descrição

Esse evento ocorre ao se deslocar o cursor do componente que indica o registro corrente para o registro anterior.

Componentes aos quais se aplica:

```
TIWDBNavigator
```

OnReceivedFile

Descrição

Esse evento ocorre quando o componente recebe os dados provenientes de um arquivo,

Componentes aos quais se aplica:

```
TIWDBFile e TIWFile
```

OnRecordChangeComplete

Descrição

Esse evento ocorre quando se alteram os valores armazenados em alguns registros de um Recordset (conjunto de registros de uma tabela ou resultantes de uma consulta SQL).

Componentes aos quais se aplica:

```
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset,
TIBStoredproc, TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery
```

OnRecordsetChangeComplete

Descrição

Esse evento ocorre quando se alteram os valores armazenados em todos os registros de um Recordset (conjunto de registrios de uma tabela ou resultantes de uma consulta SQL).

Componentes aos quais se aplica:

```
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset,
TIBStoredproc, TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery
```

OnRefresh

Descrição

Esse evento ocorre ao se alterar a propriedade DimensionMap do componente TDecisioncube ou, no caso do componente TIWDBNavigator, ao se atualizar a exibição dos registros.

Componentes aos quais se aplica:

```
TDecisionCube e TIWDBNavigator
```

OnRender

Descrição

Esse evento ocorre cada vez que o formulário for exibido como uma página HTML.

Componentes aos quais se aplica:

```
TIWForm.
```

OnRenderCell

Descrição

Esse evento ocorre cada vez que uma célula do componente for exibida em uma página HTML.

Componentes aos quais se aplica:

```
TIWDBGrid e TIWGrid.
```

OnReplace

Descrição

O evento OnReplace ocorre quando o usuário seleciona o botão Replace (Substituir) ou Replace All (Substituir Todas) em uma caixa de diálogo.

Componentes aos quais se aplica:

```
TReplaceDialog
```

OnResize

Descrição

O evento OnResize ocorre quando o componente é redimensionado durante a execução do aplicativo.

Exemplo

Se você quiser que um componente Label1 do tipo TLabel alterne sua cor entre vermelho e amarelo quando um formulário chamado Form1 é redimensionado, inclua o seguinte trecho de código:

```
procedure TForm1.FormResize(Sender: TObject);
begin
if Label1.Color = clRed then Label1.Color: = clYellow else Label1.Color: = clRed;
end;
```

Componentes aos quais se aplica:
TChart, TDBChart, TDBNavigator, TForm, TFrameTDecisionPivot,TPanel e TScrollBox

OnRestore

Descrição

O evento OnRestore ocorre quando uma aplicação minimizada restaura sua configuração default de janelas.

Componentes aos quais se aplica:
TApplicationEvents

OnRollBackTransComplete

Descrição

Esse evento ocorre após o cancelamento de uma transação.

Componentes aos quais se aplica:
TADOConnection

OnRowMoved

Descrição

O evento OnRowMoved ocorre quando o usuário movimenta uma linha com o mouse.

Exemplo

O trecho de código a seguir bloqueia movimentos posteriores de uma linha.

```
procedure TForm1.StringGrid1RowMoved(Sender: TObject; FromIndex, ToIndex: Longint);
begin
StringGrid1.Options: = StringGrid1.Options - [goRowMoving];
end;
```

Observação: Para que o evento ocorra é preciso que, inicialmente, a propriedade Options inclua a subpropriedade goRowMoving.

Componentes aos quais se aplica:
TDrawGrid e TStringGrid

OnScroll

Descrição

Esse evento ocorre quando o usuário usa uma barra de rolagem, ou quando rola o gráfico exibido em um componente.

Exemplo

O trecho de código a seguir alterna a cor de um formulário entre branco e preto, quando o usuário usa uma barra de rolagem:

```
procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
var ScrollPos: Integer);
begin
if Form1.Color = clBlack then Form1.Color: = clWhite else Form1.Color: = clBlack;
end;
```

Componentes aos quais se aplica:
TChart, TDBChart e TScrollBar

OnSectionClick

Descrição

Esse evento ocorre quando uma seção do componente é selecionada com o botão esquerdo do mouse.

Componentes aos quais se aplica:

```
THeaderControl
```

OnSectionResize

Descrição

Esse evento ocorre quando uma seção do componente é redimensionada.

Componentes aos quais se aplica:

```
THeaderControl
```

OnSectionTrack

Descrição

Esse evento ocorre quando uma seção do controle é deslocada, arrastando-se uma de suas divisórias. O procedimento associado a esse evento possui um parâmetro chamado State que indica o estado atual de arraste do componente, como indica a tabela a seguir.

Tabela de valores Para o Parâmetro State.

```
ValorSignificado
tsTrackBegin Indica o início do arraste.
tsTrackMove  Indica que a divisória continua sendo arrastada.
tsTrackEnd   Sinaliza o término do arraste.
```

Componentes aos quais se aplica:

```
THeaderControl
```

OnSelectCell

Descrição

O evento OnSelectCell ocorre quando o usuário seleciona uma célula do componente com o mouse.

Exemplo

O trecho de código a seguir faz com que uma célula selecionada em um componente StringGrid1 do tipo TStringGrid exiba os números de sua linha e coluna:

```
procedure TForm1.StringGrid1SelectCell(Sender: TObject; Col, Row: Longint;
var CanSelect: Boolean);
begin
Value: = 'Linha '+IntToStr(StringGrid1.Row)+' e Coluna '+IntToStr(StringGrid1.Col);
end;
```

Componentes aos quais se aplica:

```
TDrawGrid e TStringGrid
```

OnSetEditText

Descrição

O evento OnSetEditText ocorre quando o usuário edita o texto em uma célula do componente.

Exemplo

O trecho de código a seguir faz com que a cor das células em um componente StringGrid1 do tipo TStringGrid alterne entre vermelho e amarelo quando o usuário edita uma célula.

```
procedure TForm1.StringGrid1SetEditText(Sender: TObject; ACol,
ARow: Longint; const Value: String);
begin
if StringGrid1.Color = clYellow then StringGrid1.Color: = clRed
else StringGrid1.Color: = clYellow;
end;
```

Componentes aos quais se aplica:

TDrawGrid e TStringGrid

OnSetText

Descrição

O evento OnSetText é disparado quando se atribui um valor à propriedade Text de um campo.

Componentes aos quais se aplica:

TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField, TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField, TStringField, TTimeField, TVarBytesField e TWordField

OnShortcut

Descrição

O evento OnShortcut ocorre quando o usuário da aplicação pressiona uma tecla (este evento ocorre antes do evento OnKeyDown).

Componentes aos quais se aplica:

TApplicationEvents

OnShow

Descrição

Para componentes do tipo TForm, esse evento ocorre imediatamente antes de o formulário se tornar visível.

Para componentes do tipo TQRPreview, esse evento ocorre imediatamente antes da pré-visualização do relatório.

Exemplo

O trecho de código a seguir emite uma mensagem informando que um formulário vai se tornar visível.

```
procedure TForm1.FormShow(Sender: TObject);
begin
     ShowMessage('O formulário vai ficar visível');
end;
```

Componentes aos quais se aplica:

TForm e TQRPreview

OnShowHint

Descrição

O evento OnShowHint ocorre quando a aplicação vai exibir uma string de auxílio.

Componentes aos quais se aplica:

TApplicationEvents

OnSized

Descrição

Esse evento ocorre sempre que um componente do tipo THeader terminar de ser redimensionado.

Exemplo

O trecho de código a seguir emite uma mensagem informando que uma seção de um componente Header1 do tipo THeader foi redimensionada.

```
procedure TForm1.Header1Sized(Sender: TObject; ASection, AWidth: Integer);
begin
ShowMessage('O componente foi redimensionado');
end;
```

Componentes aos quais se aplica:

```
Theader
```

OnSizing

Descrição

Esse evento ocorre enquanto uma seção de um componente do tipo THeader está sendo redimensionada.

Exemplo

O trecho de código a seguir faz com que a cor de um componente Edit1 do tipo TEdit alterne entre vermelho e azul enquanto uma seção de um componente Header1 do tipo THeader está sendo redimensionada.

```
procedure TForm1.Header1Sizing(Sender: TObject; ASection, AWidth: Integer);
begin
if Edit1.Color = clBlue then Edit1.Color: = clRed else Edit1.Color: = clBlue;;
end;
```

Componentes aos quais se aplica:

```
THeader
```

OnSQL

Descrição

Esse evento ocorre quando se verifica a execução de comandos SQL em uma aplicação que acessa um banco de dados do Interbase através do mecanismo Interbase Express.

Componentes aos quais se aplica:

```
TIBSQLMonitor
```

OnStart

Descrição

Esse evento ocorre quando se inicia a exibição do clip de vídeo associado ao componente.

Componentes aos quais se aplica:

```
TAnimate
```

OnStartPage

Descrição

Esse evento ocorre imediatamente antes da criação de uma nova página no relatório.

Componentes aos quais se aplica:

```
TQuickReport
```

OnStateChange

Descrição

Para componentes TDataSource, o evento OnStateChange ocorre quando o valor da sua propriedade State é alterado durante a execução do aplicativo.

Para componentes TDecisionSource, o evento OnStateChange ocorre quando se altera o valor da sua propriedade DecisionCube.

Componentes aos quais se aplica:

`TDecisionSource e TDataSource`

OnStatusLineEvent

Descrição

O evento OnStatusLine ocorre quando a aplicação OLE servidora tem uma mensagem para ser exibida na linha de Status de uma aplicação OLE receptora (com um componente do tipo TOLEContainer ativo).

Exemplo

O trecho de código a seguir exibe uma mensagem de um servidor OLE em uma barra de Status representada por um componente Panel1 do tipo TPanel.

```
procedure TForm1.OleContainer1StatusLineEvent(Sender: TObject; Msg: string);
begin
Panel1.Caption: = Msg;
end;
```

Componentes aos quais se aplica:

`TOLEContainer`

OnStop

Descrição

Esse evento ocorre quando se finaliza a exibição do clip de vídeo associado ao componente.

Componentes aos quais se aplica:

`TAnimate`

OnSubmit

Descrição

Esse evento ocorre quando o componente e os dados serão enviados a partir da página HTML na qual está inserido o componente.

Componentes aos quais se aplica:

`TIWDBEdit, TIWDBFile, TIWEdit e TIWFile`

OnSummaryChange

Descrição

Esse evento ocorre quando se altera o valor da sua propriedade CurrentSum.

Componentes aos quais se aplica:

`TDecisionSource`

OnTimer

Descrição

Esse evento é usado para executar um código a intervalos regulares (esse intervalo é definido na propriedade Interval do componente).

Componentes aos quais se aplica:

TTimer

OnTopLeftChanged

Descrição

O evento OnTopLeftChanged ocorre quando uma das propriedades LeftCol ou TopRow tem o seu valor alterado.

Exemplo

O trecho de código a seguir exibe os novos valores das propriedades LeftCol e TopRow quando seus valores se alteram.

```
procedure TForm1.StringGrid1TopLeftChanged(Sender: TObject);
begin
with StringGrid1 do
MessageDlg('TopRow = '+IntToStr(TopRow)+
' e LeftCol = '+IntToStr(LeftCol), mtInformation, [mbOK],0);
end;
```

Componentes aos quais se aplica:

TDrawGrid e TStringGrid

OnUnDock

Descrição

O evento OnUnDock ocorre quando o usuário começa a arrastar um componente que está ancorado em outro objeto. Esse evento corresponde ao componente que está sendo arrastado.

Componentes aos quais se aplica:

TBitBtn, TButton, TCheckBox, TComboBox, TDBCheckBox, TDBComboBox, TDBCtrlGrid, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBMemo, TDBNavigator, TDBText, TDBRadioGroup, TDirectoryListBox, TDrawGrid, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TFrame, TGroupBox, TImage, TListBox, TMaskEdit, TMemo, TNotebook, TOLEContainer, TOutline, TDecisionPivot,TPanel, TRadioButton, TScrollBar, TScrollBox, TShape, TStringGrid e TTabSet

OnUndoZoom

Descrição

Esse evento ocorre quando o usuário desfaz a aplicação de um zoom ao gráfico exibido no componente.

Componentes aos quais se aplica:

TChart, TDBChart

OnUpdate

Descrição

O evento OnUpdate ocorre quando a lista de ações de um componente ActionList é atualizada.

Componentes aos quais se aplica:

TAction, TActionList

OnUpdateData

Descrição

O evento OnUpdateData ocorre quando o método Post ou UpdateRecord de um componente que representa um banco de dados é executado durante a execução do aplicativo.

Componentes aos quais se aplica:

```
TDataSource
```

OnUpdateError

Descrição

O evento OnUpdateError ocorre quando alguma exceção é gerada ao se utilizar o recurso de cached updates em um banco de dados acessado através do mecanismo Interbase Express.

Componentes aos quais se aplica:

```
TIBStoredProc, TIBDataset, TIBTable, TIBQuery
```

OnUpdateRecord

Descrição

O evento OnUpdateRecord ocorre quando alguma exceção é gerada ao se utilizar o recurso de cached updates em um registro de banco de dados acessado através do mecanismo Interbase Express.

Componentes aos quais se aplica:

```
TIBStoredProc, TIBDataset, TIBTable, TIBQuery
```

OnValidate

Descrição

O evento OnValidate é disparado quando o valor de um campo é modificado. Quando se faz uma alteração num controle que exibe o valor do campo, esse evento só ocorre quando o valor do campo (e não do que é exibido) é atualizado.

Componentes aos quais se aplica:

```
TAutoIncField, TBCDField, TBlobField, TBooleanField, TBytesField, TCurrencyField, TDateField,
TDateTimeField, TFloatField, TGraphicField, TIntegerField, TMemoField, TSmallintField,
TStringField, TTimeField, TVarBytesField e TWordField
```

OnWillChangeRecord

Descrição

Esse evento ocorre quando se inicia a alteração dos valores armazenados em um registro de um Recordset (conjunto de registros de uma tabela ou resultantes de uma consulta SQL).

Componentes aos quais se aplica:

```
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset,
TIBStoredproc, TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery
```

OnWillChangeRecordset

Descrição

Esse evento ocorre quando se inicia a alteração dos valores armazenados em todos os registros de um Recordset (conjunto de registros de uma tabela ou resultantes de uma consulta SQL).

Componentes aos quais se aplica:

```
TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset,
TIBStoredproc, TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery
```

OnWillConnect

Descrição

Esse evento ocorre quando se recebe um pedido para se iniciar uma conexão.

Componentes aos quais se aplica:

`TADOConnection`

OnWillExecute

Descrição

Esse evento ocorre quando o servidor de banco de dados informa a aceitação de um comando.

Componentes aos quais se aplica:

`TADOConnection`

OnZoom

Descrição

Esse evento ocorre quando o usuário aplica um zoom ao gráfico exibido no componente.

Componentes aos quais se aplica:

`TChart, TDBChart`

OnWillMove

Descrição

Esse evento ocorre imediatamente antes de se alterar o registro corrente de um Recordset (conjunto de registros de uma tabela ou resultantes de uma consulta SQL).

Componentes aos quais se aplica:

`TADOTable, TADOQuery, TADODataset, TADOStoredproc, TIBTable, TIBQuery, TIBDataset, TIBStoredproc, TClientDataset, TTable, TStoredProc e TQuery, TDecisionQuery`

Índice Remissivo

A

Abort, Método, 1230
Aborted, Propriedade, 978
AbortOnKeyViol, Propriedade, 978
AbortOnProblem, Propriedade, 978
AccuracyMethod, 979
Acesso a Bancos de Dados, Mecanismos de, 126, 394
Ações, editor de, 773
ActionCount, Propriedade, 979
ActionList, Propriedade, 979
Actions, Propriedade, 979
Activate, Método, 1230
Active, Propriedade, 980
ActiveControl, Propriedade, 980
ActiveForm, Propriedade, 981
ActiveMDIChild, Propriedade, 981
ActivePage, Propriedade, 982
Add, Método, 1230
AddChild, Método, 1231
AddChildObject, Método, 1231
AddDatabase, Método, 1231
AddDataset, Método, 1231
AddIcon, Método, 1232
AddIndex, Método, 1232
AddMasked, Método, 1232
AddObject, Método, 1232
AddPassword, Método, 1232
AddScriptFile, Método, 1233
AddSeries, Método, 1233
AddStrings, Método, 1233
AddTo Repository, caixa de diálogo, 212
AddTransaction, Método, 1233
AddXY, Método, 1234
AddY, Método, 1234
AdjustLineBreaks, função, 663
AfterCancel, Evento, 1340
AfterClose, Evento, 1340
AfterConnect, Evento, 1340
AfterConstruction, Método, 288
AfterDelete, Evento, 1340
AfterDetail, Evento, 1340
AfterDisConnect, Evento, 1340
AfterDispatch, Evento, 1340
AfterDrawValues, Evento, 1341
AfterEdit, Evento, 1341
AfterInsert, Evento, 1341
AfterOpen, Evento, 1341
AfterPost, Evento, 1341
AfterPrint, Evento, 1341
AfterScroll, Evento, 1342
Alias, criação de, 134
AliasName, Propriedade, 982
Align, Propriedade, 982
AlignButton, Propriedade, 984
Alignment, Propriedade, 984
AlignToBand, Propriedade, 985
Allocation, Propriedade, 986
AllocBy, Propriedade, 986
AllowAllUp, Propriedade, 986
AllowDelete, Propriedade, 987
AllowGrayed, Propriedade, 987
AllowInPlace, Propriedade, 987
AllowInsert, Propriedade, 988
AllowPaning, Propriedade, 988
AllowResize, Propriedade, 989
AllowSinglePoint, Propriedade, 989
AllowZoom, Propriedade, 989
Ambiente de Desenvolvimento Integrado, 10
  barra de menus do, 11
  barra de títulos do, 11
  caixa de ferramentas, 11
  janela principal do, 11
  janelas do, 11
  paleta de componentes, 11
Ancoragem de janelas, recurso de, 36
AnsiCompareStr, função, 663
AnsiCompareText, função, 664
AnsiExtractQuotedStr, função, 664
AnsiLowerCase, função, 664
AnsiPos, função, 664
AnsiQuotedStr, função, 664
AnsiStrComp, função, 671
AnsiStrIComp, função, 671
AnsiStrLComp, função, 671
AnsiStrLower, função, 671
AnsiStrPos, função, 672
AnsiStrRScan, função, 672
AnsiStrScan, função, 672
AnsiStrUpper, função, 672
AnsiUpperCase, função, 664
Aplicações CGI, criação de, 770
Aplicações MDI, 745
Aplicações Multicamadas, 728
Aplicações Websnap, 791
Aplicações, internacionalização de, 758
Aplicações, planejamento de, 74
Append, função, 658
Append, Método, 1234
AppendRecord, Método, 1234
ApplyRange, Método, 1234
ApplyUpdates, Método, 1235
Arc, Método, 1235
ArcCos, função, 822
ArcCosH, função, 822
ArcSin, função, 822
ArcSinH, função, 822
ArcTan2, função, 822
ArcTanH, função, 822
Aritméticos, operadores, 51
Arquivos, manipulação de, 658
ArrangeIcons, Método, 1235
ArrowKeys, Propriedade, 989
As, operador, 295
AsBoolean, Propriedade, 990
AsDateTime, Propriedade, 990
AsFloat, Propriedade, 990
AsInteger, Propriedade, 991
Assign, Método, 1235
AssignFile, procedimento, 659
AssignValue, Método, 1236
Associate, Propriedade, 991
AsString, Propriedade, 991
AsText, Propriedade, 992
AtLeast, Método, 1236
AutoActivate, Propriedade, 992
AutoCalcFields, Propriedade, 993
AutoDisplay, Propriedade, 993
AutoEdit, Propriedade, 994
AutoEnable, Propriedade, 994
AutoMerge, Propriedade, 994
AutoOpen, Propriedade, 995
AutoPopup, Propriedade, 995
AutoRewind, Propriedade, 995
AutoScroll, Propriedade, 996
AutoSelect, Propriedade, 996
AutoSize, Propriedade, 996
AxisVisible, Propriedade, 997

B

Back, Método, 1237
BackColor, Propriedade, 997
BackgroundColor, Propriedade, 997
BackImage, Propriedade, 998
Bancos de Dados,
  Cliente-Servidor, 572
  conceitos e definição, 127
  mecanismos de acesso a, 126, 394
  metadados de, 589
Band Style Editor, caixa de diálogo, 240
BatchMove, Método, 1237
BeforeCancel, Evento, 1342
BeforeClose, Evento, 1342
BeforeConnect, Evento, 1342
BeforeConstruction, Método, 288
BeforeDelete, Evento, 1342
BeforeDetail, Evento, 1343
BeforeDisconnect, Evento, 1343
BeforeDispatch, Evento, 1343
BeforeDrawValues, Evento, 1343
BeforeEdit, Evento, 1343
BeforeInsert, Evento, 1343
BeforeOpen, Evento, 1343
BeforePost, Evento, 1344
BeforePrintt, Evento, 1344
BeforeRefresh, Evento, 1344
BeforeScroll, Evento, 1344
BeginDoc, Método, 1237
BeginDrag, Método, 1237
BeginTrans, Método, 1238
BeginUpdate, Método, 1238
BevelInner, Propriedade, 998
BevelOuter, Propriedade, 999
BevelWidth, Propriedade, 999
Bitmap, Propriedade, 1000
BkColor, Propriedade, 1000
BlendColor, Propriedade, 1000
BlobSize, Propriedade, 1000
Blocos de Comandos, 46, 54, 55
BOF, Propriedade, 1001
BookMarkValid, Método, 1238
BorderIcons, Propriedade, 1001
BorderStyle, Propriedade, 1001
BorderWidth, Propriedade, 1002
Botão de comando, componente, 122
BottomAxis, Propriedade, 1003
BoundsRect, Propriedade, 1003
Break, Propriedade, 1004
BringToFront, Método, 1239
BroadCast, Método, 1239
Brush, Propriedade, 1004
BrushCopy, Método, 1239
ButtonAutoSize, Propriedade, 1005
ButtonHeight, Propriedade, 1005
ButtonSpacing, Propriedade, 1005
ButtonWidth, Propriedade, 1005
BytesRcvd, Propriedade, 1006
BytesSent, Propriedade, 1006
BytesTotal, Propriedade, 1006

C

CachedSize, Propriedade, 1006
CachedUpdates, Propriedade, 1006
CachedUpdates, uso de, 495
Caixas de diálogo, criação, 118

Caixas de diálogo, definição, 118
Caixas de diálogo, exibição, 120
Caixas de diálogo, modal, 120
Caixas de diálogo, não-modal, 120
Calculated, Propriedade, 1007
Call, Método, 1239
Campos calculados, definição de, 196, 445
Campos Lookup, definição de, 449
CanAutoSize, Método, 1240
Cancel, Método, 1240
Cancel, Propriedade, 1007
CancelDispatch, Método, 1240
Canceled, Propriedade, 1007
CancelEvents, Método, 1240
CancelRange, Método, 1240
CancelUpdates, Método, 1241
CanFocus, Método, 1241
CanModify, Propriedade, 1008
Canvas, Propriedade, 1008
Capabilities, Propriedade, 1009
Capacity, Propriedade, 1009
Caption, Propriedade, 1010
CaptionColor, Propriedade, 1010
CaptionFont, Propriedade, 1010
Cascade, Método, 1241
Category, Propriedade, 1011
Ceil, função, 822
CellRect, Método, 1241
Cells, Propriedade, 1011
Center, Propriedade, 1011
Change, Método, 1242
ChangedCount, Propriedade, 1012
ChangedTableName, Propriedade, 1012
ChangeLevelBy, Método, 1242
CharCase, Propriedade, 1012
Chave Primária, 438
CheckActive, Método, 1242
CheckBrowseMode, Método, 1242
CheckClosed, Método, 1242
CheckDatabaseInList, Método, 1243
CheckDatabasename, Método, 1243
Checked, Propriedade, 1013
CheckInactive, Método, 1243
CheckInTransaction, Método, 1243
CheckNotInTransaction, Método, 1243
CheckOpen, Método, 1244
CheckValidStatement, Método, 1244
Classes,
  EAbort, 830
  EAbstractError, 830
  EAccessViolation, 830
  EArrayError, 831
  EAssertionFailed, 831
  EBitsError, 831
  EBrokerException, 832
  ECacheError, 832
  EClassNotFound, 832
  ECommonCalendarError, 833
  EComponentError, 833
  EControlC, 833
  EConvertError, 834
  EDatabaseError, 835
  EdateTimeError, 836
  EDBClient, 834
  EDBEditError, 834
  EDBEngineError, 835
  EdimensionMapError, 836
  EDimIndexError, 836
  EdivByZero, 837
  EDSWriter, 835
  EExternal, 837
  EExternalException, 837
  EFCreateError, 837
  EFilerError, 838
  EFOpenError, 838
  EHeapException, 838
  EInOutError, 839
  EIntError, 840
  EIntfCastError, 840
  EIntOverFlow, 840
  EInvalidArgument, 841
  EInvalidCast, 841
  EInvalidGraphic, 841
  EInvalidGraphicOperation, 842
  EInvalidGridOperation, 842
  EInvalidImage, 842
  EInvalidOp, 843
  EInvalidOperation, 843
  EInvalidPointer, 843
  EListError, 844
  ElowCapacityError, 844
  EMathError, 844
  EMCIDeviceError, 845
  EMenuError, 845
  EMonthCalError, 845
  EndResultSet, 846
  EOLECtrlError, 846
  EOLEError, 846
  EOLEException, 847
  EOLESysError, 847
  EOSError, 847
  EOutlineError, 848
  EOutOfMemory, 848
  EOutOfResources, 848
  EOverflow, 849
  EPackageError, 849
  EParseError, 849
  EPrinter, 850
  EPrivilege, 850
  EPropertyError, 850
  EPropReadOnly, 851
  EPropWriteOnly, 851
  EreadError, 852
  EReconcileError, 852
  ERegistryException, 852
  EResNotFound, 853
  ESocketConnectionError, 853
  ESocketError, 853
  EStackoverflow, 854
  EStreamError, 854
  EStringListError, 854
  EThread, 855
  ETreeViewError, 855
  EUnderflow, 855
  EUpdateError, 856
  EVariantError, 856
  EWin32Error, 857
  EWriteError, 857
  Exception, 298
  EZeroDivide, 857
  TAction, 858
  TActionList, 858
  TADOCommand, 525, 859
  TADOConnection, 512, 859
  TADODataSet, 527, 860
  TADOQuery, 530, 860
  TADOStoredProc, 861
  TADOTable, 529, 862
  TAnimate, 863
  TApplication, 863
  TApplicationEvents, 863
  TAutoIncField, 864
  TBaseReport, 864
  TBatchMove, 865
  TBCDField, 865
  TBDEDataset, 416
  TBevel, 866
  TBitBtn, 866
  TBitmap, 626, 866
  TBlobField, 867
  TBlobStream, 867
  TBooleanField, 868
  TBrush, 868
  TButton, 868
  TBytesField, 869
  TCanvas, 335
  TChart, 870
  TChartSeries, 871
  TCheckBox, 871
  TClientDataSet, 872
  TClientSocket, 873
  TClipboard, 873
  TColorDialog, 873
  TComboBox, 874
  TComponent, 874
  TControlScrollBar, 875
  TCorbaConnection, 875
  TCorbaDatamodule, 729
  TCurrencyField, 875
  TCustomADODataSet, 518, 876
  TCustomConnection, 411
  TCustomSQLDataset, 544
  TDataBase, 419, 876
  TDataSet, 395
  TDataSetPageProducer, 877
  TDataSetProvider, 877
  TDataSetTableProducer, 877
  TDataSource, 878
  TDateField, 878
  TDateTimeField, 878
  TDBChart, 879
  TDBCheckBox, 879
  TDBComboBox, 880
  TDBCtrlGrid, 880
  TDBDataset, 418
  TDBEdit, 881
  TDBGrid, 881
  TDBImage, 882
  TDBListBox, 882
  TDBLookupComboBox, 883
  TDBLookupListBox, 883
  TDBMemo, 884
  TDBMemoBuf, 884
  TDBNavigator, 885
  TDBRadioGroup, 885
  TDBRichEdit, 886
  TDBText, 886
  TDCOMConnection, 887
  TDDEClientConv, 887
  TDDEClientItem, 887
  TDDEServerConv, 888
  TDDEServerItem, 888
  TDecisionCube, 888
  TDecisionGraph, 888
  TDecisionGrid, 889
  TDecisionPivot, 890
  TDecisionQuery, 890
  TDecisionSource, 890
  TDirectoryListBox, 891
  TDrawGrid, 891
  TDriveComboBox, 892
  TEdit, 892
  TField, 893
  TFieldDef, 893
  TFileListBox, 893
  TFilterComboBox, 894
  TFindDialog, 894
  TFloatField, 895
  TFont, 895
  TFontDialog, 895
  TForm, 68, 896
  TGraphic, 896
  TGraphicsField, 897
  TGroupBox, 897
  THeader, 897
  THeaderControl, 898
  THotKey, 898
  THTTPFile, 907
  TIBClientDataSet, 899
  TIBCustomDataSet, 558
  TIBDatabase, 552, 900
  TIBDatabaseInfo, 900
  TIBDataSet, 561, 901
  TIBEvents, 901
  TIBQuery, 567, 902
  TIBSQL, 902
  TIBSQLMonitor, 903
  TIBStoredProc, 903
  TIBTable, 563, 904
  TIBTransaction, 558, 905
  TIBUpdateSQL, 568, 905
  TIcon, 905
  TImage, 906
  TImageList, 906
  TIniFile, 907
  TIntegerField, 907
  TIWAppForm, 908

TIWApplet, 908
TIWApplication, 908
TIWButton, 909
TIWCheckBox, 909
TIWComboBox, 910
TIWControl, 910
TIWDBCheckBox, 910
TIWDBComboBox, 911
TIWDBEdit, 911
TIWDBFile, 911
TIWDBGrid, 912
TIWDBImage, 912
TIWDBListBox, 913
TIWDBLookupComboBox, 913
TIWDBLookupListBox, 913
TIWDBMemo, 914
TIWDBNavigator, 914
TIWDBText, 915
TIWEdit, 915
TIWFile, 915
TIWForm, 916
TIWGrid, 916
TIWImage, 917
TIWImageFile, 917
TIWLabel, 917
TIWLink, 918
TIWList, 918
TIWListBox, 918
TIWMemo, 919
TIWRectangle, 919
TIWRegion, 920
TIWTimer, 920
TIWTreeView, 920
TIWURL, 921
TLabel, 921
TList, 305, 921
TListBox, 922
TListView, 922
TMaskEdit, 923
TMediaPlayer, 924
TMemo, 924
TMemoBuf, 925
TMemoField, 925
TMenuItem, 925
TMetafile, 926
TMIDASConnection, 926
TMIDASPageProducer, 926
TMTSDatamodule, 729
TNMDayTime, 927
TNMEcho, 927
TNMFinger, 927
TNMFTP, 928
TNMHTTP, 928
TNMMSG, 928
TNMMSGServ, 929
TNMNNTP, 929
TNMPOP3, 929
TNMSMTP, 930
TNMSTRM, 930
TNMSTRMServ, 931
TNMTime, 931
TNMUDP, 931
TNMURL, 932
TNMUUProcessor, 930
Tnotebook, 932
TObject, 287
TOLEContainer, 933
TOpenDialog, 933
TOpenPictureDialog, 933
TOutline, 934
TOutlineNode, 934
TPageControl, 935
TPageProducer, 935
TPaintBox, 935
TPanel, 936
TPen, 341, 936
TPicture, 937
TPopupMenu, 937
TPrintDialog, 937
TPrinter, 637, 938
TPrinterSetupDialog, 938
TProgressBar, 939
TProvider, 939
TQRDBText, 939
TQRLabel, 940
TQRMemo, 940
TQRPreview, 940
TQRPrinter, 941
TQRShape, 941
TQRSysData, 941
TQuery, 434, 942
TQueryTableProducer, 942
TQueue, 942
TQuickRep, 943
TRadioButton, 943
TRadioGroup, 944
TRDSConnection, 517, 944
TRegistry, 945
TRegistryIniFile, 945
TRemoteServer, 945
TReplaceDialog, 946
TRichEdit, 946
TRpBarsBase, 946
TRpBaseComponent, 947
TRpComponent, 947
TRpRender, 948
TRpRenderCanvas, 948
TRpRenderStream, 948
TRvCustomConnection, 949
TRvDatsetConnection, 949
TRvNDRWriter, 949
TRvProject, 950
TRvQueryConnection, 954
TRvRenderHTML, 951
TRvRenderPDF, 951
TRvRenderPreview, 951
TRvRenderPrinter, 952
TRvRenderRTF, 953
TRvRenderText, 954
TRvSystem, 954
TRvTableConnection, 955
TSaveDialog, 955
TSavePictureDialog, 955
TScreen, 956
TScrollBar, 956
TScrollBox, 957
TServerSocket, 957
TSession, 422, 958
TShape, 958
TSimpleDataset, 548, 958
TSimpleObjectBroker, 959
TSmallIntField, 960
TSocketConnection, 960
TSpeedButton, 960
TSQLConnection, 540, 961
TSQLData Set, 544, 961
TSQLMonitor, 962
TSQLQuery, 547, 962
TSQLStoredProc, 963
TSQLTable, 546, 964
TStack, 964
TStatusBar, 965
TStoredProc, 595, 965
TStringField, 966
TStringGrid, 966
TStringList, 966
TStrings, 967
TTabbedNotebook, 967
TTabControl, 968
TTable, 428, 968
TTabSet, 969
TTabSheet, 969
TThread, 708, 970
TTimeField, 970
TTimer, 970
TToolBar, 971
TTrackBar, 971
TTreeView, 972
TUpdateSQL, 437, 500, 972
TUpDown, 973
TVarBytesField, 973
TWebDispatcher, 974
TWordField, 974
TXMLBroker, 974
wEInterpreterError, 839
Classes e objetos, noções de, 63
Classes,
  conceito de, 64
  herança de, 69, 275
  implementação de, 272
  Métodos Abstratos, 284
  Métodos de, 65, 285
  Métodos e campos,
    privados, 71, 280
    protegidos, 71, 280
    públicos, 70, 279
  Métodos Estáticos, Virtuais e Dinâmicos, 282
  Propriedades, 285
  Sobreposição de Métodos, 281
  visibilidade de Métodos e campos, 279
ClassInfo, Método 288, Método, 1244
ClassName, Método 289, 1244
ClassNameLs, Método 289, 1245
ClassParent, Método 289, 1245
ClassType, Método 290, 1245
CleanUp, Método, 1246
CleanupInstance, Método 290, 1246
Clear, Método, 1246
ClearFields, Método, 1247
ClearSelection, Método, 1247
Click, Método, 1247
ClientDataset, componente, 732
ClientHandle, Propriedade, 1014
ClientHeight, Propriedade, 1014
ClientOrigin, Propriedade, 1014
ClientRect, Propriedade, 1015
ClientToScreen, Método, 1248
ClientWidth, Propriedade, 1015
ClipRect, 1016
Close, Método, 1248
CloseDatabase, Método, 1249
CloseDataSets, Método, 1249
CloseDialog, Método, 1249
CloseLink, Método, 1250
CloseQuery, Método, 1250
CloseUp, Método, 1250
Code Explorer, janela 35
Códigos, Editor de, 12
Col, Propriedade, 1016
ColCount, Propriedade, 1016
Collapse, Método, 1251
Collate, Propriedade, 1016
Color, Propriedade, 1017
ColorEachPoint, Propriedade, 1019
ColoredButtons, Propriedade, 1019
ColorToRGBString, Método, 1251
Cols, Propriedade, 1019
ColumnMarginInches, Propriedade, 1019
ColumnMarginMM, Propriedade, 1020
Columns, Propriedade, 1020
ColWidths, Propriedade, 1020
COM, criação de objeto, 716
COM, tecnologia, 716, 729
Comandos, blocos de, 46, 54, 55
Comentários, 46
Command, Propriedade, 1021
CommandText, Propriedade, 1021
CommandTimeOut, Propriedade, 1021
CommandType, Propriedade, 1022
Commit, Método, 1251
CommitRetaining, Método, 1251
CommitTrans, Método, 1251
CommitUpdates, Método, 1252
CommonAVI, 1022
CompareStr, função, 665
CompareText, função, 665
ComponentCount, Propriedade, 1023
Componente,
  Botão de Comando, 122
  ClientDataset, 159
  ClientDataset, principais Métodos, 179
  ClientDataset, principais Propriedades, 178

DatasetProvider, 159
DatasetTableProducer, 611
Datasource, 154, 180
DBComboBox, 168
DBEdit, 145, 164
DBGrid, 210
DBNavigator, 205
MaskEdit, 167
RvProject, 236
Shape, 343, 611,
SimpleDataset, 180
Componentes,
conceito de, 15
criação de Eventos, 360
criação de Propriedades, 352
criação de, 348
definição de, 15, 328, 329
nomenclatura de, 76
paleta de, 12
redimensionando, 93
renomeando, 92
reposicionando, 92
templates de, 200
ComponentIndex, Propriedade, 1023
Components, Propriedade, 1024
Concat, função, 665
Condicionais, Estruturas, 54, 55
ConfirmDelete, Propriedade, 1024
Conjuntos, 47
operações sobre, 48
Connect, Método, 1252
Connected, Propriedade, 1024
Connection, Propriedade, 1025
ConnectionString, Propriedade, 1025
ConnectMode, Propriedade, 1025
Construtores, Métodos, 277
ContainsControl, Método, 1252
Content, Método, 1252
Context, Propriedade, 1026
ControlAtPos, Método, 1252
ControlCount, Propriedade, 1026
Controles ActiveX, criação de, 379
Controles e componentes, definição de, 15
Controls, Propriedade, 1027
ControlType, Propriedade, 1027
ConvertDlgHelp, Propriedade, 1028
Copies, Propriedade, 1028
Copy, função, 665
CopyParams, Método, 1253
CopyRect, Método, 1253
CopyToClipboard, Método, 1253
CorbaObject, Método, 1253
CosH, função, 823
Cotan, função, 823
Count, Propriedade, 1028
Create Database, caixa de diálogo, 132
Create, Método, 288, 299, 1253
CreateDatabase, Método, 1255
CreateField, Método, 1255
CreateFmt, Método 299, 1255
CreateFmtHelp, Método, 300, 1256
CreateHelp, Método, 300, 1256
CreateIndexFile, Método, 1257
CreateNew, Método, 1257
CreateRes, Método, 300, 1257
CreateResFmt, Método, 301, 1257
CreateResFmtHelp, Método, 301, 1258
CreateSize, Método, 1258
CreateTable, Método, 1258
Ctl3D, Propriedade, 1029
CurrebtPage, 1030
Currency, Propriedade, 1029
CurrentMemory, Propriedade, 1030
CurrentSum, Propriedade, 1030
CurrToStr, função, 665
Cursor, Propriedade, 1030
CursorPosChanged, Método, 1259
Cursors, Propriedade, 1031
CursorXPos, 1032
CursorYPos, 1032
CustomColors, Propriedade, 1033
Customize, Caixa de diálogo, 14
CutToClipboard, Método, 1259
CycleToRad, função, 823
D
Data Connection, caixa de diálogo, 238
Data View Dictionary, 238
Data, Propriedade, 1033
Database Desktop, utilitário 133
Database, Propriedade, 1033
DatabaseCount, Propriedade, 1033
DatabaseDisconnected, Evento, 1344
DatabaseDisconnecting, Evento, 1344
DatabaseFree, Evento, 1345
DatabaseName, Propriedade, 1034
Databases, Propriedade, 1034
DataField, Propriedade, 155, 1034
Datamodule, definição de, 174
DataSet, Propriedade, 1035
DataSetCount, Propriedade, 1035
DatasetProvider, componente, 730
DataSets, Propriedade, 1035
DatasetTableProducer Componente, 782
DataSize, Propriedade, 1036
DataSource, componente, 154
DataSource, Propriedade, 1036
DataSource, Propriedade, 155
DataType, Propriedade, 1037
Dataview, Propriedade, 241
DateToStr, função, 666
DBCombobox, componente, 168
DBCombobox, componente, Propriedades, 168
DBEdit, componente, 155, 164
DBExpress Connection Editor, caixa de diálogo, 153
DBExpress, mecanismo de acesso a dados, 152
DBFileName, Propriedade, 1037
DBHandle, Propriedade, 1037
DBLocale, Propriedade, 1037
DBNavigator, componente, 205
DBSQLDialect, Propriedade, 1038
DCOM, Tecnologia, 729
DCOMConnection, Componente, 731
DDEConv, Propriedade, 1038
DDEItem, Propriedade, 1038
DDEService, Propriedade, 1039
DDETopic, Propriedade, 1039
DecisionCube, Propriedade, 1039
DecisionSource, Propriedade, 1039
Default, Propriedade, 1040
DefaultColWidth, Propriedade, 1040
DefaultDatabase, Propriedade, 1041
DefaultDrawing, Propriedade, 1041
DefaultExt, Propriedade, 1041
DefaultHandler, Método, 290
DefaultIndex, Propriedade, 1042
DefaultRowHeight, Propriedade, 1042
DefaultTransaction, Propriedade, 1042
DegToRad, função, 823
Delete, função, 666
Delete, Método, 1259
DeleteIndex, Método, 1260
DeleteSQL, Propriedade, 1042
DeleteTable, Método, 1260
DescriptionAvailable, Método, 1260
Desenvolvimento, ferramentas de, 4
DesignReport, Método, 1260
Destination, Propriedade, 1043
Destroy, Método 291, 1260
Destroy, Método, 290
DestroyComponents, Método, 1261
Destroying, Método, 1261
Destrutores, Métodos, 278
Device, Propriedade, 1043
DeviceID, Propriedade, 1044
DeviceType, Propriedade, 1044
DimensionMap, Propriedade, 1044
Directory, Propriedade, 1045
DirLabel, Propriedade, 1045
DirList, Propriedade, 1045
DisableAlign, Método, 1261
DisableControls, Método, 1261
DisableIfNoHandler, Propriedade, 1046
DisConnect, Método, 1261
Dispatch, Método 290, 1262
Display, Propriedade, 1046
DisplayFormat, Propriedade, 182, 1047
DisplayLabel, Propriedade, 1047
DisplayName, Propriedade, 1047
DisplayPrintDialog, Propriedade, 1047
DisplayRect, Propriedade, 1048
DisplayText, Propriedade, 1048
DisplayValue, Propriedade, 1049
DisplayWidth, Propriedade, 1049
DitherBackground, Propriedade, 1049
DLLs, criação de, 650
DocBackColor, Propriedade, 1050
DocForeColor, Propriedade, 1050
DockClientCount, Propriedade, 1050
DockClients, Propriedade, 1050
DockRect, Propriedade, 1050
DockSite, Propriedade, 1051
DocLinkColor, Propriedade, 1051
DoHint, Método, 1262
DoTerminate, Método, 1262
Down, Propriedade, 1051
Drag-And-Drop, operações de, 751
DragCursor, Propriedade, 1052
Dragging, Método, 1262
Dragging, Propriedade, 1052

DragLock, Método, 1263
DragMode, Propriedade, 1052
DragUnLock, Método, 1263
Draw, Método, 1263
DrawFocusRect, Método, 1263
DrawingStyle, Propriedade, 1053
DrawOverlay, Método, 1263
Drive, Propriedade, 1053
DriverName, Propriedade, 1054
DropConnections, Método, 1263
DropDatabase, Método, 1264
DropDown, Método, 1264
DropDownAlign, Propriedade, 1054
DropDownCount, Propriedade, 1055
DropDownWidth, Propriedade, 1055
DropTarget, Propriedade, 1055

E

EAbort, Classe, 830
EAbstractError, Classe, 830
EAccessViolation, Classe, 830
EArrayError, Classe, 831
EAssertionFailed, Classe, 831
EBitsError, Classe, 831
EBrokerException, Classe, 832
ECacheError, Classe, 832
EClassNotFound, Classe, 832
ECommonCalendarError, Classe, 833
EComponentError, Classe, 833
EControlC, Classe, 833
EConvertError, Classe, 834
EDatabaseError, Classe, 835
EDateTimeError, Classe, 836
EDBClient, Classe, 834
EDBEditError, Classe, 834
EDBEngineError, Classe, 835
EDimensionMapError, Classe, 836
EDimIndexError, Classe, 836
Edit Tab Order, Caixa de diálogo, 171
Edit, Método, 1264
EditFormat, Propriedade, 182, 1056
EditKey, Método, 1264
EditMask, Propriedade, 165, 1056
EditMaskPtr, Propriedade, 1057
EditMode, Propriedade, 1058
Editor de Código, 12
EditorMode, Propriedade, 1058
EditRangeEnd, Método, 1265
EditRangeStart, Método, 1265
EditText, Propriedade, 1058
EdivByZero, Classe, 837
EDSWriter, Classe, 835
EExternal, Classe, 837
EExternalException, Classe, 837
EFCreateError, Classe, 837
EFilerError, Classe, 838
EFOpenError, Classe, 838
EHeapException, Classe, 838
EInOutError, Classe, 839
EInterpreterError, Classe, 839
EIntError, Classe, 840
EIntfCastError, Classe, 840
EIntOverFlow, Classe, 840
EInvalidArgument, Classe, 841
EInvalidCast, Classe, 841
EInvalidGraphic, Classe, 841
EInvalidGraphicOperation, Classe, 842
EInvalidGridOperation, Classe, 842
EInvalidImage, Classe, 842
EInvalidOp, Classe, 843
EInvalidOperation, Classe, 843
EInvalidPointer, Classe, 843
Eject, Método, 1265
EListError, Classe, 844
Ellipse, Método, 1265
ElowCapacityError, Classe, 844
EMathError, Classe, 844
EMCIDeviceError, Classe, 845
EMenuError, Classe, 845
EMonthCalError, Classe, 845
Empty, Propriedade, 1059
EmptyTable, Método, 1266
EnableAlign, Método, 1266
EnableControls, Método, 1266
Enabled, Propriedade, 1059
EnabledButtons, Propriedade, 1059
EnableOpenBtn, Propriedade, 1060
EnablePrintBtn, Propriedade, 1060
EnableSaveBtn, Propriedade, 1060
EndDoc, Método, 1266
EndDrag, Método, 1266
EndMargin, Propriedade, 1060
EndPos, Propriedade, 1061
EndUpdate, Método, 1267
EnoResultSet, Classe, 846
EOF, Propriedade, 1061
EOLECtrlError, Classe, 846
EOLEError, Classe, 846
EOLEException, Classe, 847
EOLESysError, Classe, 847
EOSError, Classe, 847
EOutlineError, Classe, 848
EOutOfMemory, Classe, 848
EOutOfResources, Classe, 848
EOverflow, Classe, 849
EPackageError, Classe, 849
EParseError, Classe, 849
EPrinter, Classe, 850
EPrivilege, Classe, 850
EPropertyError, Classe, 850
EPropReadOnly, Classe, 851
EPropWriteOnly, Classe, 851
EraseSection, Método, 1267
EReadError, Classe, 852
EReconcileError, Classe, 852
ERegistryException, Classe, 852
EResNotFound, Classe, 853
Error, Propriedade, 1061
ErrorCode, Propriedade, 1062
ErrorCount, Propriedade, 1062
ErrorMessage, Propriedade, 1062
Errors, Propriedade, 1063
ESocketConnectionError, Classe, 853
ESocketError, Classe, 853
EStackOverflow, Classe, 854
EStreamError, Classe, 854
EStringListError, Classe, 854
Estruturas Condicionais, 54, 55
Estruturas de repetição, 54, 57
EThread, Classe, 855
ETreeViewError, Classe, 855
EUnderflow, Classe, 855
EUpdateError, Classe, 856
EVariantError, Classe, 856
Evento,
  AfterCancel, 1340
  AfterClose, 1340
  AfterConnect, 1340
  AfterDelete, 1340
  AfterDetail, 1340
  AfterDisConnect, 1340
  AfterDispatch, 1340
  AfterDrawValues, 1341
  AfterEdit, 1341
  AfterInsert, 1341
  AfterOpen, 1341
  AfterPost, 1341
  AfterPrint, 1341
  AfterScroll, 1342
  BeforeCancel, 1342
  BeforeClose, 1342
  BeforeConnect, 1342
  BeforeDelete, 1342
  BeforeDetail, 1343
  BeforeDisconnect, 1343
  BeforeDispatch, 1343
  BeforeDrawValues, 1343
  BeforeEdit, 1343
  BeforeInsert, 1343
  BeforeOpen, 1343
  BeforePost, 1344
  BeforePrintt, 1344
  BeforeRefresh, 1344
  BeforeScroll, 1344
  DatabaseDisconnected, 1344
  DatabaseDisconnecting, 1344
  DatabaseFree, 1345
  OnActionExecute, 1345
  OnActionUpdate, 1345
  OnActivate, 1345
  OnActiveControlChange, 1345
  OnActiveFormChange, 1346
  OnAfterAdd, 1346
  OnAfterClose, 1346
  OnAfterDraw, 1346
  OnAfterOpen, 1346
  OnAfterPivot, 1346
  OnAfterPrint, 1346
  OnAfterRender, 1347
  OnApply, 1347
  OnBeforeAdd, 1347
  OnBeforeClose, 1347
  OnBeforeOpen, 1347
  OnBeforePivot, 1347
  OnBeforePrint, 1348
  OnBeginRetrieval, 1348
  OnBeginTransComplete, 1348
  OnCalcFields, 1348
  OnCancel, 1348
  OnChange, 1348
  OnChanging, 1349
  OnClearValues, 1349
  OnClick, 1349
  OnClickAxis, 1350
  OnClickLegend, 1350
  OnClose, 1350
  OnCloseQuery, 1351
  OnColEnter, 1351
  OnColExit, 1352
  OnCollapse, 1352
  OnColumnClick, 1352
  OnColumnMoved, 1352
  OnCompare, 1353
  OnConnect, 1353
  OnConnectComplete, 1353
  OnConnectionFailed, 1353
  OnCreate, 1353
  OnDataChange, 1354
  OnDblClick, 1354
  OnDeActivate, 1354
  OnDefault, 1355
  OnDelete, 1355
  OnDeletion, 1355
  OnDesignerSave, 1355
  OnDesignerSaveAs, 1355
  OnDesignerShow, 1355
  OnDestroy, 1355
  OnDisConnect, 1356
  OnDockDrop, 1356
  OnDockOver, 1356
  OnDragDrop, 1357
  OnDragOver, 1357
  OnDrawCell, 1357
  OnDrawItem, 1358
  OnDrawTab, 1358
  OnDropDown, 1359
  OnEdit, 1359
  OnEditError, 1359
  OnEndDock, 1359
  OnEndDrag, 1359
  OnEndOfRecordset, 1360
  OnEndPage, 1360
  OnEndRetrieval, 1360
  OnEnter, 1360
  OnEventAlert, 1360
  OnException, 1361
  OnExecute, 1361
  OnExecuteComplete, 1361
  OnExecuteMacro, 1361
  OnExit, 1361
  OnExpand, 1361
  OnFetchComplete, 1362
  OnFetchProgress, 1362
  OnFilter, 1362
  OnFilterRecord, 1362

OnFind, 1362
OnFirst, 1363
OnGetEditMask, 1363
OnGetEditText, 1363
OnGetErrorResponse, 1363
OnGetResponse, 1363
OnGetText, 1364
OnHelp, 1364
OnHide, 1364
OnHint, 1364
OnHTMLTag, 1364
OnIdle, 1365
OnIdleTimer, 1365
OnInfoMessage, 1365
OnInsert, 1365
OnKeyDown, 1365
OnKeyPress, 1366
OnKeyUp, 1366
OnLast, 1366
OnLogin, 1366
OnLowCapacity, 1366
OnMeasureItem, 1367
OnMeasureTab, 1367
OnMessage, 1368
OnMinimize, 1368
OnMouseDown, 1368
OnMouseUp, 1369
OnMouseWheel, 1369
OnMouseWheelUp, 1369
OnMoveComplete, 1370
OnNeedData, 1370
OnNewDimensions, 1370
OnNewPage, 1370
OnNewRecord, 1370
OnNext, 1370
OnNotify, 1371
OnOpen, 1371
OnPaint, 1371
OnPassword, 1371
OnPokeData, 1372
OnPopup, 1372
OnPost, 1372
OnPostClick, 1372
OnPostError, 1372
OnPreview, 1372
OnPrint, 1373
OnPrintFooter, 1373
OnPrintHeader, 1373
OnPrior, 1373
OnReceivedFile, 1373
OnRefresh, 1374
OnRender, 1374
OnRenderCell, 1374
OnReplace, 1374
OnResize, 1374
OnRestore, 1375
OnRowMoved, 1375
OnScroll, 1375
OnSectionClick, 1376
OnSectionResize, 1376
OnSectionTrack, 1376
OnSelectCell, 1376
OnSetEditText, 1376
OnSetText, 1377
OnShortcut, 1377
OnShow, 1377
OnShowHint, 1377
OnSized, 1378
OnSizing, 1378
OnSQL, 1378
OnStart, 1378
OnStartPage, 1378
OnStateChange, 1379
OnStatusLineEvent, 1379
OnStop, 1379
OnSubmit, 1379
OnSummaryChange, 1379
OnTimer, 1380
OnTopLeftChanged, 1380
OnUnDock, 1380
OnUndoZoom, 1380
OnUpdate, 1380
OnUpdateData, 1381
OnUpdateError, 1381
OnUpdateRecord, 1381
OnValidate, 1381
OnWillConnect, 1382
OnWillExecute, 1382
OnWillMove, 1382
OnZoom, 1382
Eventos, definição de, 18
Events, Propriedade, 1063
EWin32Error, Classe, 857
EWriteError, Classe, 857
Exceções,
  conceito de, 296
  tratamento de, 296
Exception, Classe, 298
Exchange, Método, 1267
Exclusive, Propriedade, 1063
ExecProc, Método, 1267
ExecQuery, Método, 1268
ExecSQL, Método, 1268
Execute, Método, 1268
ExecuteAction, Método, 1269
ExecuteMacro, Método, 1269
ExecuteMacroLines, Método, 1269
ExecuteReport, Método, 1270
ExeName, Propriedade, 1063
Exists, Propriedade, 1064
Expand, Método, 1270
Expanded, Propriedade, 1064
ExtendSelect, Propriedade, 1064
EZeroDivide, Classe, 857

F

Ferramentas, caixa de, 11
Ferramentas RAD, conceito, 4
FieldAddress, Método 291, 1270
FieldByName, Método, 1270
FieldClass, Propriedade, 1064
FieldCount, Propriedade, 1065
FieldDefs, Propriedade, 1065
FieldList, Propriedade, 1065
FieldName, Propriedade, 1065
FieldNo, Propriedade, 1066
Fields Editor, definição, 160
Fields, Propriedade, 1066
FieldValues, Propriedade, 1066
FileEdit, Propriedade, 1067
FileEditStyle, Propriedade, 1067
FileExtension, Propriedade, 1067
FileGetDate, Método, 659
FileList, Propriedade, 1068
FileLoad, Método, 1271
FileName, Propriedade, 1068
FileOpen, Método, 659
FileSearch, função, 659
FileSetAttr, função, 660
FileSetDate, função, 660
FileType, Propriedade, 1069
FileWrite, função, 660
FillRect, Método, 1271
Filter, Propriedade, 1070
Filtered, Propriedade, 1070
FilterIndex, Propriedade, 1070
FindClose, função, 660
FindComponent, Método, 1271
FindDatabase, Método, 1271
FindField, Método, 1272
FindFirst, função, 660
FindKey, Método, 1272
FindNearest, Método, 1272
FindNext, função, 661
FindNextPage, Método, 1272
FindText, Propriedade, 1071
FindTransaction, Método, 1273
First, Método, 1273
FirstIndex, Propriedade, 1071
FirstPage, Propriedade, 1071
FixedColor, Propriedade, 1072
FixedCols, Propriedade, 1072
FixedRows, Propriedade, 1072
FloatToStr, função, 666
FloodFill, Método, 1273
FocusControl, Método, 1273
FocusControl, Propriedade, 1073
Focused, Método, 1274
Focused, Propriedade, 1073
Font, Propriedade, 85, 1073
Fonts, Propriedade, 1074
FooterBand, Propriedade, 1074
ForceDirectories, função, 661
ForceNewPage, Propriedade, 1075
FormatChars, Propriedade, 1075
Formats, Propriedade, 1075
FormCount, Propriedade, 1076
Forms, Propriedade, 1076
FormStyle, Propriedade, 1076
Formulários, alinhando componentes, 97
Formulários, definição de, 15
Formulários, inserindo componentes, 90, 96
Formulários, manipulação de, 84
Formulários, selecionando componentes, 92
Formulários, templates de, 210
Frame, Propriedade, 1077
FrameCount, Propriedade, 1077
FrameHeight, Propriedade, 1077
FrameRect, Método, 1274
Frames, Propriedade, 1078
FrameWidth, Propriedade, 1077
Free, Método 290, 291, 1274
FreeBookMark, Método, 1275
FreeInstance, Método, 290, 291, 1275
FreeNotification, Método, 1275
FreeOnTerminated, Propriedade, 1078
Frequency, Propriedade, 1078
Frexp, função, 823
FromPage, Propriedade, 1078
FullCollapse, Método, 1275
FullExpand, Método, 1275
FullPath, Propriedade, 1079
Funções, 59, 61
  para manipulação de arquivos, 62

G

GetAliasNames, Método, 1276
GetAliasParams, Método, 1276
GetAsHandle, Método, 1276
GetBitmap, Método, 1276
GetBookmark, Método, 1277
GetChildren, Método, 1277
GetComponent, Método, 1277
GetCurrentDir, função, 661
GetData, Método, 1277
GetDatabaseNames, Método, 1278
GetDataItem, Método, 1278
GetDeltaPacket, Método, 1278
GetDetailLinkFields, Método, 1278
GetDetailSQL, Método, 1278
GetDir, função, 661
GetDriverNames, Método, 1279
GetDriverParams, Método, 1279
GetErrorCount, Método, 1279
GetFieldNames, Método, 1279
GetFirstChild, Método, 1280
GetFormImage, Método, 1280
GetHotSpot, Método, 1280
GetIcon, Método, 1280
GetImageBitmap, Método, 1280
GetIndexForPage, Método, 1281
GetIndexNames, Método, 1281
GetItem, Método, 1281
GetItemPath, Método, 1281

GetLastChild, Método, 1281
GetMaskBitmap, Método, 1282
GetNamePath, Método, 1282
GetNextChild, Método, 1282
GetParentComponent, Método, 1282
GetPassword, Método, 1282
GetPrevChild, Método, 1283
GetPrinter, Método, 1283
GetProcedureNames, Método, 1283
GetResults, Método, 1283
GetSelTextBuf, Método, 1283
GetSQL, Método, 1284
GetStoredProcNames, Método, 1284
GetTableNames, Método, 1284
GetTabOrderList, Método, 1285
GetText, Método, 1285
GetTextBuf, Método, 1285
GetTextLen, Método, 1285
GetXMLRecords, Método, 1286
Glyph, Propriedade, 211, 1079
GotoBookmark, Método, 1286
GotoCurrent, Método, 1286
GotoKey, Método, 1286
GotoNearest, Método, 1287
GradToRad, função, 823
Graphic, Propriedade, 1080
GridHeight, Propriedade, 1080
GridLineWidth, Propriedade, 1080
GridWidth, Propriedade, 1081
GroupIndex, Propriedade, 1081
Grupos de Projeto, Manipulação de, 32

H

Handle, Propriedade, 1081
HandleAllocated, Método, 1287
HandleException, Método, 1287
HandleIsShared, Propriedade, 1084
HandleNeeded, Método, 1287
HasFormat, Método, 1288
Hash, Método, 1288
HasItems, Propriedade, 1084
HasProvider, Propriedade, 1085
Head1Font, Propriedade, 1085
Head2Font, Propriedade, 1085
Head3Font, Propriedade, 1085
Head4Font, Propriedade, 1086
Head5Font, Propriedade, 1086
Head6Font, Propriedade, 1086
HeaderBand, Propriedade, 1086
Height, Propriedade, 1087
Help, criação de arquivos de, 248
HelpCommand, Método, 1288
HelpContext, Método, 1289
HelpContext, Propriedade, 302, 1087
HelpFile, Propriedade, 1088
HelpJump, Método, 1289
HelpKeyWord, Propriedade, 1088
HelpType, Propriedade, 1088
Hide, Método, 1289
HideDragImage, Método, 1290
HideScrollBars, Propriedade, 1089
HideSelection, Propriedade, 1089
Hint, Propriedade, 1089
HintColor, Propriedade, 1090
HintPause, Propriedade, 1090
Hints, Propriedade, 206, 1091
HistoryList, Propriedade, 1091
HorizAxis, Propriedade, 1091
HorzScrollBar, Propriedade, 1092
Host, Propriedade, 1092
HotKey, Propriedade, 1092
HTML, formulários, 777
HTMLDoc, Propriedade, 780
HTMLFile, Propriedade, 780
Hypot, função, 824

I

IBConsole, utilitário, 131
Icon, Propriedade, 89, 1093
IdleTimer, Propriedade, 1093
ImageIndex, Propriedade, 1093
Images, Propriedade, 1094
ImageType, Propriedade, 1094
Impressão, técnicas de, 637
Increment, Propriedade, 1094
Indent, Propriedade, 1095
Index, Propriedade, 1095
IndexDefs, Propriedade, 1095
IndexFieldCount, Propriedade, 1096
IndexFieldNames, Propriedade, 1096
IndexFields, Propriedade, 1096
IndexName, Propriedade, 1096
IndexOf, Método, 1290
IndexOfObject, Método, 1290
Índices, criação de, 141, 183
InheritsFrom, Método, 293
InitialDir, Propriedade, 1097
InitInstance, Método 292, 1291
InitInstance, Método, 292
InPlaceActive, Propriedade, 1097
Input Mask Editor, caixa de diálogo, 166
Insert,
  comando, 583
  função, 666
  Método, 170, 1291
InsertComponent, Método, 1292
InsertControl, Método, 1292
InsertIcon, Método, 1292
InsertMasked, Método, 1292
InsertObject, Método, 1293
InsertRecord, Método, 1293
InsertSQL, Propriedade, 1097
InstanceSize, Método 292, 1293
IntegralHeight, Propriedade, 1098
Interbase Interactive SQL, utilitário, 576
Interbase Server Manager, utilitário, 573
Interbase,
  acesso via BDE, 594
  acesso via DBExpress, 605
  acesso via Interbase Express, 600
    Alterando um Usuário, 575
  Cadastrando um Novo Usuário, 574
  Criando um Banco de Dados no, 576
  Criando uma Tabela no, 579
  linguagem de codificação do, 587
  Removendo um Usuário, 575
  Tipos de dados do, 578
Interval, Propriedade, 1098
IntPower, função, 824
Intransaction, Propriedade, 1098
Intraweb, tecnologia, 806
IntToStr, função, 666
InValidate, Método, 1293
InvalidKeys, Propriedade, 1099
Is, operador, 294
IsA, Método, 1293
IsDelimiter, função, 666
IsEmpty, Método, 1294
IsIndexField, Propriedade, 1099
IsLocal, Método, 1294
IsMasked, Propriedade, 1099
IsNull, Propriedade, 1100
IsPathDelimiter, função, 666
IsReadOnly, Propriedade, 1100
IsSQLBased, Propriedade, 1100
IsValidChar, Método, 1294
IsVisible, Propriedade, 1101
ItemAtPos, Método, 1294
ItemCount, Propriedade, 1101
ItemHeight, Propriedade, 1101
ItemIndex, Propriedade, 1102
ItemRect, Método, 1295
Items,
  objeto, 169
  objeto, Métodos, 169
  objeto, Propriedades, 169
  Propriedade, 169, 1102
ItemSeparator, Propriedade, 1103

J

Janela, garantindo a visibilidade de uma, 37

K

KeepConnections, Propriedade, 1103
KeyExclusive, Propriedade, 1104
KeyField, Propriedade, 1104
KeyFieldCount, Propriedade, 1104
KeyPreview, Propriedade, 1104
KeyValue, Propriedade, 1105
KeyViolCount, Propriedade, 1105
KeyViolTableName, Propriedade, 1105
Kind, Propriedade, 1106

L

Label, componente, 155
LargeChange, Propriedade, 1107
LargeImages, Propriedade, 1107
Last, Método, 1295
LastDelimiter, função, 667
LastPage, Propriedade, 1107
Layout, Propriedade, 1108
Left, Propriedade, 1108
LeftAxis, Propriedade, 1109
LeftCol, Propriedade, 1109
LeftMarginInches, Propriedade, 1109
LeftMarginMM, Propriedade, 1109
Length, função, 667
Length, Propriedade, 1110
Level, Propriedade, 1110
Lines, Propriedade, 1110
LineTo, Método, 1295
Linguagem SQL, 216
LinkBand, Propriedade, 1111
List, Propriedade, 1111
ListField, Propriedade, 1111
ListSource, Propriedade, 1112
Load, Método, 1295
LoadFromFile, Método, 170, 1296
LoadFromStream, Método, 1296
LoadMemo, Método, 1296
LoadPicture, Método, 1297
Local, Propriedade, 1112
Locale, Propriedade, 1112
LocalIP, Propriedade, 1112
Locate, Método, 1297
Locked, Propriedade, 1113
LockTable, Método, 1297
Log10, função, 824
Log2, função, 824
Lógicos, operadores, 59
LoginPrompt, Propriedade, 1113
LogN, função, 824
LookupDisplay, Propriedade, 1113

LookupField, Propriedade, 1114
LookupSource, Propriedade, 1114
LowerCase, função, 667

M

MainForm, Propriedade, 1114
Mapeamento, modos de, 670
Mappings, Propriedade, 1115
Margin, Propriedade, 1115
MarginBotton, Propriedade, 1116
MarginLeft, Propriedade, 1116
MarginRight, Propriedade, 1116
MarginTop, Propriedade, 1116
Máscaras, definição de, 164
Mask, Propriedade, 1117
Masked, Propriedade, 1117
MaskEdit, componente, 167
Master, Propriedade, 1117
MasterFields, Propriedade, 1118
MasterSource, Propriedade, 1118
Max, função, 824
Max, Propriedade, 1118
MaxCells, Propriedade, 1118
MaxFontSize, Propriedade, 1119
MaxIntValue, função, 825
MaxLength, Propriedade, 1119
MaxPage, Propriedade, 1119
MaxSummaries, Propriedade, 1120
MaxValue, função, 825
MaxValue, Propriedade, 1120
MDI, aplicações, 745
MDIChildCount, Propriedade, 1120
MDIChildren, Propriedade, 1121
Mean, função, 825
MeanAndStdDev, função, 825
Menu, Propriedade, 1121
Menus pop-up,
  associando Eventos a itens de, 115
  criando itens de, 110
  criando submenus de, 112
  inclusão de, 109
Menus,
  associando Eventos a itens de, 114
  barra de, 11
  criando itens de, 105
  criando separador de itens de, 107
  criando teclas aceleradoras para itens de, 108
  definição de, 102
  editor de, 103
  inclusão de, 102
  mesclando, 747
Merge, Método, 1297
Message, Propriedade, 302, 1121
MessageBox, Método, 1298
Metafile, Propriedade, 1122
MethodAddress, Método 293, 1298
MethodName, Método, 293, 1298
Método,
  Abort, 1230
  Activate, 1230
  Add, 1230
  AddChild, 1231
  AddChildObject, 1231
  AddDatabase, 1231
  AddDataset, 1231
  AddIcon, 1232
  AddIndex, 1232
  AddMasked, 1232
  AddObject, 1232
  AddPassword, 1232
  AddScriptFile, 1233
  AddSeries, 1233
  AddStrings, 1233
  AddTransaction, 1233
  AddXY, 1234
  AddY, 1234
  Append, 1234
  AppendRecord, 1234
  ApplyRange, 1234
  ApplyUpdates, 1235
  Arc, 1235
  ArrangeIcons, 1235
  Assign, 1235
  AssignValue, 1236
  AtLeast, 1236
  Back, 1237
  BatchMove, 1237
  BeginDoc, 1237
  BeginDrag, 1237
  BeginTrans, 1238
  BeginUpdate, 1238
  BookMarkValid, 1238
  BringToFront, 1239
  BroadCast, 1239
  BrushCopy, 1239
  Call, 1239
  CanAutoSize, 1240
  Cancel, 1240
  CancelDispatch, 1240
  CancelEvents, 1240
  CancelRange, 1240
  CancelUpdates, 1241
  CanFocus, 1241
  Cascade, 1241
  CellRect, 1241
  Change, 1242
  ChangeLevelBy, 1242
  CheckActive, 1242
  CheckBrowseMode, 1242
  CheckClosed, 1242
  CheckDatabaseInList, 1243
  CheckDatabasename, 1243
  CheckInactive, 1243
  CheckInTransaction, 1243
  CheckNotInTransaction, 1243
  CheckOpen, 1244
  CheckValidStatement, 1244
  ClassInfo, 288, 1244
  ClassName, 289, 1244
  ClassNameLs, 289, 1245
  ClassParent, 289, 1245
  ClassType, 290, 1245
  CleanUp, 1246
  CleanupInstance, 290, 1246
  Clear, 1246
  ClearFields, 1247
  ClearSelection, 1247
  Click, 1247
  ClientToScreen, 1248
  Close, 1248
  CloseDatabase, 1249
  CloseDataSets, 1249
  CloseDialog, 1249
  CloseLink, 1250
  CloseQuery, 1250
  CloseUp, 1250
  Collapse, 1251
  ColorToRGBString, 1251
  Commit, 1251
  CommitRetaining, 1251
  CommitTrans, 1251
  CommitUpdates, 1252
  Connect, 1252
  ContainsControl, 1252
  Content, 1252
  ControlAtPos, 1252
  CopyParams, 1253
  CopyRect, 1253
  CopyToClipboard, 1253
  CorbaObject, 1253
  Create, 288, 299, 1253
  CreateDatabase, 1255
  CreateField, 1255
  CreateFmt, 299, 1255
  CreateFmtHelp, 300, 1256
  CreateForm, 1256
  CreateHelp, 300, 1256
  CreateIndexFile, 1257
  CreateNew, 1257
  CreateRes, 300, 1257
  CreateResFmt, 301, 1257
  CreateResFmtHelp, 301, 1258
  CreateSize, 1258
  CreateTable, 1258
  CursorPosChanged, 1259
  CutToClipboard, 1259
  Delete, 1259
  DeleteIndex, 1260
  DeleteTable, 1260
  DescriptionAvailable, 1260
  DesignReport, 1260
  Destroy, 291, 1260
  DestroyComponents, 1261
  Destroying, 1261
  DisableAlign, 1261
  DisableControls, 1261
  DisConnect, 1261
  Dispatch, 290, 1262
  DoHint, 1262
  DoTerminate, 1262
  Dragging, 1262
  DragLock, 1263
  DragUnLock, 1263
  Draw, 1263
  DrawFocusRect, 1263
  DrawOverlay, 1263
  DropConnections, 1263
  DropDatabase, 1264
  DropDown, 1264
  Edit, 1264
  EditKey, 1264
  EditRangeEnd, 1265
  EditRangeStart, 1265
  Eject, 1265
  Ellipse, 1265
  EmptyTable, 1266
  EnableAlign, 1266
  EnableControls, 1266
  EndDoc, 1266
  EndDrag, 1266
  EndUpdate, 1267
  EraseSection, 1267
  Exchange, 1267
  ExecProc, 1267
  ExecQuery, 1268
  ExecSQL, 1268
  Execute, 1268
  ExecuteAction, 1269
  ExecuteMacro, 1269
  ExecuteMacroLines, 1269
  ExecuteReport, 1270
  Expand, 1270
  FieldAddress, 291, 1270
  FieldByName, 1270
  FileLoad, 1271
  FillRect, 1271
  FindComponent, 1271
  FindDatabase, 1271
  FindField, 1272
  FindKey, 1272
  FindNearest, 1272
  FindNextPage, 1272
  FindTransaction, 1273
  First, 1273
  FloodFill, 1273
  FocusControl, 1273
  Focused, 1274
  FrameRect, 1274
  Free, 290, 1274
  FreeBookMark, 1275
  FreeInstance, 290, 1275
  FreeNotification, 1275
  FullCollapse, 1275
  FullExpand, 1275
  GetAliasNames, 1276
  GetAliasParams, 1276
  GetAsHandle, 1276
  GetBitmap, 1276
  GetBookmark, 1277
  GetChildren, 1277
  GetComponent, 1277
  GetData, 1277
  GetDatabaseNames, 1278

GetDataItem, 1278
GetDeltaPacket, 1278
GetDetailLinkFields, 1278
GetDetailSQL, 1278
GetDriverNames, 1279
GetDriverParams, 1279
GetErrorCount, 1279
GetFieldNames, 1279
GetFirstChild, 1280
GetFormImage, 1280
GetHotSpot, 1280
GetIcon, 1280
GetImageBitmap, 1280
GetIndexForPage, 1281
GetIndexNames, 1281
GetItem, 1281
GetItemPath, 1281
GetLastChild, 1281
GetMaskBitmap, 1282
GetNamePath, 1282
GetNextChild, 1282
GetParentComponent, 1282
GetPassword, 1282
GetPrevChild, 1283
GetPrinter, 1283
GetProcedureNames, 1283
GetResults, 1283
GetSelTextBuf, 1283
GetSQL, 1284
GetStoredProcNames, 1284
GetTableNames, 1284
GetTabOrderList, 1285
GetText, 1285
GetTextBuf, 1285
GetTextLen, 1285
GetXMLRecords, 1286
GotoBookmark, 1286
GotoCurrent, 1286
GotoKey, 1286
GotoNearest, 1287
HandleAllocated, 1287
HandleException, 1287
HandleNeeded, 1287
HasFormat, 1288
Hash, 1288
HelpCommand, 1288
HelpContext, 1289
HelpJump, 1289
Hide, 1289
HideDragImage, 1290
IndexOf, 1290
IndexOfObject, 1290
InheritsFrom, 293, 1290
InitInstance, 292, 1291
Insert, 170, 1291
InsertComponent, 1292
InsertControl, 1292
InsertIcon, 1292
InsertMasked, 1292
InsertObject, 1293
InsertRecord, 1293
InstanceSize, 292, 1293
InValidate, 1293
IsA, 1293
IsEmpty, 1294
IsLocal, 1294
IsValidChar, 1294
ItemAtPos, 1294
ItemRect, 1295
Last, 1295
LineTo, 1295
Load, 1295
LoadFromFile, 170, 1296
LoadFromStream, 1296
LoadMemo, 1296
LoadPicture, 1297
Locate, 1297
LockTable, 1297
Merge, 1297
MessageBox, 1298
MethodAddress, 293, 1298
MethodName, 293, 1298
Minimize, 1298
MouseToCell, 1299
Move, 1299
MoveBy, 1299
MoveTo, 1299
NewInstance, 292, 1300
NewPage, 1300
Next, 1300
NonExistent, 1301
NormalizeTopMosts, 1301
OLE., 1301
OleObjAllocated, 1301
Open, 1301
OpenDatabase, 1302
OpenIndexFile, 1302
OpenLink, 1302
Overlay, 1303
Pack, 1303
ParamByName, 1303
PasteFromClipboard, 1303
Pause, 1303
PauseOnly, 1304
Peek, 1304
Perform, 1304
Pie, 1305
Play, 1305
PokeData, 1305
PokeDataLines, 1305
Polygon, 1306
Polyline, 1306
Pop, 1306
Pop-up, 1306
Post, 1306
Prepare, 1307
Preview, 1307
Previous, 1307
Print, 1307
Prior, 1308
ProcessMessages, 1308
Push, 1308
QueueEvents, 1308
Read, 1309
ReadBool, 1309
ReadInteger, 1309
ReadSection, 1309
ReadSectionValues, 1309
ReadString, 1310
Realign, 1310
Rectangle, 1310
Refresh, 1310
RegisterChanges, 1311
RegisterEvents, 1311
Release, 1311
ReleaseHandle, 1311
ReleasePalette, 1311
Remove, 1312
RemoveAllPassword, 1312
RemoveAllSeries, 1312
RemoveComponent, 1312
RemoveDatabase, 1313
RemoveDatabases, 1313
RemovePassword, 1313
RemoveSeries, 1313
RemoveTransaction, 1314
Repaint, 1314
ReplaceIcon, 1314
ReplaceMasked, 1314
RequestData, 1315
Reset, 1315
ResourceLoad, 1315
Restore, 1315
RestoreTopMosts, 1316
Resume, 1316
Rewind, 1316
Rollback, 1317
RollBackRetaining, 1317
RoundRect, 1317
Run, 1317
Save, 1317
SaveToBitmapFile, 1318
SaveToFile, 170, 1318
SaveToMetaFile, 1318
SaveToMetaFileEnh, 1319
SaveToStream, 1319
ScaleBy, 1319
ScreenToClient, 1319
ScrollBy, 1320
ScrollInView, 1320
Seek, 1321
SelectAll, 1321
SelectNext, 1321
SelectNextPage, 1322
SendToBack, 1322
SeriesCount, 1322
SetAsHandle, 1322
SetBounds, 1323
SetComponent, 1323
SetData, 1323
SetFields, 1323
SetFocus, 1324
SetKey, 1324
SetLink, 1324
SetParams, 1324
SetPrincipal, 1324
SetPrinter, 1325
SetRange, 1325
SetRangeEnd, 1325
SetRangeStart, 1325
SetSelTextBuf, 1326
SetTabFocus, 1326
SetText, 1326
SetTextBuf, 1326
SetUpdateState, 1327
Show, 1327
ShowCubeDialog, 1327
ShowDragImage, 1328
ShowException, 1328
ShowModal, 1328
StartRecording, 1329
StartTransaction, 1329
Step, 1329
Stop, 1329
StretchDraw, 1330
Suspend, 1330
Synchronize, 1330
Terminate, 1330
TestConnected, 1331
TextHeight, 1331
TextOut, 1331
TextRect, 1331
TextWidth, 1331
Tile, 1332
Truncate, 1332
UnlockTable, 1332
UnMerge, 1332
UnPrepare, 1333
UnRegisterChanges, 1333
UnRegisterEvents, 1333
Update, 1333
UpdateCursrPos, 1334
UpdateRecord, 1334
ValidateEdit, 1334
Write, 1334
WriteBCDData, 1335
WriteBlobData, 1335
WriteBool, 1335
WriteBoolData, 1335
WriteCurrData, 1335
WriteDateTime, 1336
WriteFloatData, 1336
WriteIntData, 1336
WriteInteger, 1366
WriteNullData, 1336
WriteStrData, 1337
WriteString, 1337
ZoomToFit, 1337
ZoomToWidth, 1337
Métodos,
definição de, 17
sobrecarga de, 322
sobreposição de, 281
Min, função, 825
Min, Propriedade, 1122
MinFontSize, Propriedade, 1122
Minimize, Método, 1298
MinIntValue, função, 825
MinPage, Propriedade, 1123
MinValue, função, 825
MinValue, Propriedade, 1123
MKDir, função, 662
ModalResult, Propriedade, 1123
Mode, Propriedade, 1124
Modified, Propriedade, 1126
Modifiers, Propriedade, 1126
ModifySQL, Propriedade, 1127
Monochrome, Propriedade, 1127
MouseToCell, Método, 1299

Move, Método, 1299
MoveBy, Método,1299
MovedCount, Propriedade, 1127
MoveTo, Método, 1299
Multicamadas, aplicações, 728
Multiline, Propriedade, 1128
MultiSelect, Propriedade, 1128

N

Name, Propriedade, 1128
NameList, Propriedade, 1129
Navegação, componentes e Métodos de, 473
NetFileDir, Propriedade, 1129
New Connection, caixa de diálogo, 153
New Items, caixa de diálogo, 29, 214
NewInstance, Método, 292, 1300
NewPage, Método, 1300
Next, Método, 1300
NonExistent, Método, 1301
Norm, função, 826
NormalizeTopMosts, Método, 1301
Notify, Propriedade, 1129
NotifyValue, Propriedade, 1130
NumGlyphs, Propriedade, 1130

O

ObjClass, Propriedade, 1131
ObjDoc, Propriedade, 1131
Object Inspector, 11, 18
Object Inspector,
definindo procedimentos associados a Eventos, 20
exibindo Propriedades no, 331
Object Treeview, janela, 12
Objects, Propriedade, 1131
Objetos, definição de, 63
ObjItem, Propriedade, 1131
OEMConvert, Propriedade, 1132
OLE, Método, 1301
OleObjAllocated, Método, 1301
OnActionExecute, Evento, 1345
OnActionUpdate, Evento, 1345
OnActivate, Evento, 1345
OnActiveControlChange, Evento, 1345
OnActiveFormChange, Evento, 1346
OnAfterAdd, Evento, 1346
OnAfterClose, Evento, 1346
OnAfterDraw, Evento, 1346
OnAfterOpen, Evento, 1346
OnAfterPivot, Evento, 1346
OnAfterPrint, Evento, 1346
OnAfterRender, Evento, 1347
OnApply, Evento, 1347
OnBeforeAdd, Evento, 1347
OnBeforeClose, Evento, 1347
OnBeforeOpen, Evento, 1347
OnBeforePivot, Evento, 1347
OnBeforePrint, Evento, 1348
OnBeginRetrieval, Evento, 1348
OnBeginTransComplete, Evento, 1348
OnCalcFields, Evento, 1348
OnCancel, Evento, 1348
OnChange, Evento, 1348
OnChanging, Evento, 1349
OnClearValues, Evento, 1349
OnClick, Evento, 1349
OnClickAxis, Evento, 1350
OnClickLegend, Evento, 1350
OnClose, Evento, 1350
OnCloseQuery, Evento, 1351
OnColEnter, Evento, 1351
OnColExit, Evento, 1352
OnCollapse, Evento, 1352
OnColumnClick, Evento, 1352
OnColumnMoved, Evento, 1352
OnCommitTransComplete, Evento, 1353
OnCompare, Evento, 1353
OnConnect, Evento, 1353
OnConnectComplete, Evento, 1353
OnConnectionFailed, Evento, 1353
OnConnectionRequired, Evento, 1353
OnCreate, Evento, 1353
OnDataChange, Evento, 1354
OnDblClick, Evento, 1354
OnDeActivate, Evento, 1354
OnDefault, Evento, 1355
OnDelete, Evento, 1355
OnDeletion, Evento, 1355
OnDesignerSave, Evento, 1355
OnDesignerSaveAs, Evento, 1355
OnDesignerShow, Evento, 1355
OnDestroy, Evento, 1355
OnDialectDowngradeWarning, Evento, 1356
OnDisConnect, Evento, 1356
OnDockDrop, Evento, 1356
OnDockOver, Evento, 1356
OnDragDrop, Evento, 1357
OnDragOver, Evento, 1357
OnDrawCell, Evento, 1357
OnDrawItem, Evento, 1358
OnDrawTab, Evento, 1358
OnDropDown, Evento, 1359
OnEdit, Evento, 1359
OnEditError, Evento, 1359
OnEndDock, Evento, 1359
OnEndDrag, Evento, 1359
OnEndOfRecordset, Evento, 1360
OnEndPage, Evento, 1360
OnEndRetrieval, Evento, 1360
OnEnter, Evento, 1360
OnEventAlert, Evento, 1360
OnException, Evento, 1361
OnExecute, Evento, 1361
OnExecuteComplete, Evento, 1361
OnExecuteMacro, Evento, 1361
OnExit, Evento, 1361
OnExpand, Evento, 1361
OnFetchComplete, Evento, 1362
OnFetchProgress, Evento, 1362
OnFilter, Evento, 1362
OnFilterRecord, Evento, 1362
OnFind, Evento, 1362
OnFirst, Evento, 1363
OnGetEditMask, Evento, 1363
OnGetEditText, Evento, 1363
OnGetErrorResponse, Evento, 1363
OnGetResponse, Evento, 1363
OnGetText, Evento, 1364
OnHelp, Evento, 1364
OnHide, Evento, 1364
OnHint, Evento, 1364
OnHTMLTag, Evento, 1364
OnIdle, Evento, 1365
OnIdleTimer, Evento, 1365
OnInfoMessage, Evento, 1365
OnInsert, Evento, 1365
OnKeyDown, Evento, 1365
OnKeyPress, Evento, 1366
OnKeyUp, Evento, 1366
OnLast, Evento, 1366
OnLogin, Evento, 1366
OnLowCapacity, Evento, 1366
OnMeasureItem, Evento, 1367
OnMeasureTab, Evento, 1367
OnMessage, Evento, 1368
OnMinimize, Evento, 1368
OnMouseDown, Evento, 1368
OnMouseMove, Evento, 1368
OnMouseUp, Evento, 1369
OnMouseWheel, Evento, 1369
OnMouseWheelDown, Evento, 1369
OnMouseWheelUp, Evento, 1369
OnMoveComplete, Evento, 1370
OnNeedData, Evento, 1370
OnNewDimensions, Evento, 1370
OnNewPage, Evento, 1370
OnNewRecord, Evento, 1370
OnNext, Evento, 1370
OnNotify, Evento, 1371
OnOpen, Evento, 1371
OnPaint, Evento, 1371
OnPassword, Evento, 1371
OnPokeData, Evento, 1372
OnPopup, Evento, 1372
OnPost, Evento, 1372
OnPostClick, Evento, 1372
OnPostError, Evento, 1372
OnPreview, Evento, 1372
OnPrint, Evento, 1373
OnPrintFooter, Evento, 1373
OnPrintHeader, Evento, 1373
OnPrior, Evento, 1373
OnReceivedFile, Evento, 1373
OnRecordChangeComplete, Evento, 1372
OnRecordsetChangeComplete, Evento, 1374
OnRefresh, Evento, 1374
OnRender, Evento, 1374
OnRenderCell, Evento, 1374
OnReplace, Evento, 1374
OnResize, Evento, 1374
OnRestore, Evento, 1375
OnRollBackTransComplete, Evento, 1375
OnRowMoved, Evento, 1375
OnScroll, Evento, 1375
OnSectionClick, Evento, 1376
OnSectionResize, Evento, 1376
OnSectionTrack, Evento, 1376
OnSelectCell, Evento, 1376
OnSetEditText, Evento, 1376
OnSetText, Evento, 1377
OnShortcut, Evento, 1377
OnShow, Evento, 1377
OnShowHint, Evento, 1377
OnSized, Evento, 1378
OnSizing, Evento, 1378
OnSQL, Evento, 1378
OnStart, Evento, 1378
OnStartPage, Evento, 1378
OnStateChange, Evento, 1379
OnStatusLineEvent, Evento, 1379
OnStop, Evento, 1379
OnSubmit, Evento, 1379
OnSummaryChange, Evento, 1379
OnTimer, Evento, 1380
OnTopLeftChanged, Evento, 1380
OnUnDock, Evento, 1380
OnUndoZoom, Evento, 1380
OnUpdate, Evento, 1380
OnUpdateData, Evento, 1381
OnUpdateError, Evento, 1381
OnUpdateRecord, Evento, 1381

OnValidate, Evento, 1381
OnWillChangeRecord, Evento, 1381
OnWillChangeRecordset, Evento, 1381
OnWillConnect, Evento, 1382
OnWillExecute, Evento, 1382
OnWillMove, Evento, 1382
OnZoom, Evento, 1382
Open, Método, 1301
Open, Propriedade, 1132
OpenDatabase, Método, 1302
OpenIndexFile, Método, 1302
OpenLink, Método, 1302
Operadores aritméticos, 51
Operadores de Conversão de tipos, 294
Operadores Lógicos, 58
Operadores relacionais, 56
Operation, Propriedade, 1132
Options, Propriedade, 1133
Orientation, Propriedade, 1139
OriginalException, Propriedade, 1140
OutlineStyle, Propriedade, 1141
Output Options, caixa de diálogo, 241
Overlay, Método, 1303
Overload, Propriedade, 1141
Owner, Propriedade, 1142
Pc “
Pack, Método, 1303
PageCount, Propriedade, 1142
PageHeight, Propriedade, 1142
PageIndex, Propriedade, 1143
PageNumber, Propriedade, 1143
PageProducer Componente, 780
Pages, Propriedade, 1144
PageSize, Propriedade, 1144
PageWidth, Propriedade, 1144
Palette, Propriedade, 1145
PanelBorder, Propriedade, 1145
PanelCount, Propriedade, 1145
PanelHeight, Propriedade, 1146
PanelIndex, Propriedade, 1146
Panels, Propriedade, 1146
PanelWidth, Propriedade, 1146
Paragraph, Propriedade, 1147
ParamBindMode, Propriedade, 1147
ParamByName, Método, 1303
ParamCheck, Propriedade, 1147
ParamCount, Propriedade, 1148
Parameters, Propriedade, 1148
Parâmetros Default, definição, 324
Params, Propriedade, 1148
Parent, Propriedade, 1149
ParentChart, Propriedade, 1149
ParentColor, Propriedade, 1149
ParentCtl3D, Propriedade, 1150
ParentFont, Propriedade, 1150
ParentShowHint, Propriedade, 1151
PasswordChar, Propriedade, 1151
PasteFromClipboard, Método, 1303
Pause, Método, 1303
PauseOnly, Método, 1304
Peek, Método, 1304
Pen, Propriedade, 1152
PenPos, Propriedade, 1152
Perform, Método, 1304
Picture, Propriedade, 1152
PictureClosed, Propriedade, 1152
PictureLeaf, Propriedade, 1153
PictureMinus, Propriedade, 1153
PictureOpen, Propriedade, 1154
PicturePlus, Propriedade, 1154
Pie, Método, 1305
Pixels, Propriedade, 1154
PixelsPerInch, Propriedade, 1155
PlainText, Propriedade, 1155
Play, Método, 1305
PokeData, Método, 1305
PokeDataLines, Método, 1305
Poly, função, 826
Polygon, Método, 1306
Polyline, Método, 1306
Pop, Método, 1306
PopNStdDev, função, 826
PopNVariance, função, 826
Pop-up, Método, 1306
PopupComponent, Propriedade, 1155
PopupMenu, Propriedade, 1156
Port, Propriedade, 1156
Pos, função, 667
Position, Propriedade, 1156
Post, Método, 1306
PostMessage, Propriedade, 1157
Power, função, 826
Precision, Propriedade, 1158
Prepare, Método, 1307
Prepared, Propriedade, 1158
Preview, Método, 1307
Previous, Método, 1307
PreviousError, Propriedade, 1158
Print, Método, 1307
PrintBefore, Propriedade, 1158
PrinterIndex, Propriedade, 1159
PrinterOK, Propriedade, 1159
Printers, Propriedade, 1159
Printing, Propriedade, 1159
PrintMask, Propriedade, 1160
PrintRange, Propriedade, 1160
PrintScale, Propriedade, 1160
PrintToFile, Propriedade, 1161
Prior, Método, 1308
PrivateDir, Propriedade, 1161
ProblemCount, Propriedade, 1162
ProblemTableName, Propriedade, 1162
Procedimentos associados a Eventos, 20, 71
Procedimentos, 59, 60
ProcedureName, Propriedade, 1162
ProcessMessages, Método, 1308
Programação Orientada a Objetos, filosofia da, 269
Programação procedural, 264
ProjectFile, Propriedade, 1163
Projeto de Aplicação,
  arquivo de, 23
  conceito de, 22
  fechando, 31
  grupos de, 32
  iniciando um novo, 28
  salvando, 29
Propriedade,
  Aborted, 978
  AbortOnKeyViol, 978
  AbortOnProblem, 978
  AccuracyMethod, 979
  ActionCount, 979
  ActionList, 979
  Actions, 979
  Active, 980
  ActiveControl, 980
  ActiveForm, 981
  ActiveMDIChild, 981
  ActivePage, 982
  AliasName, 982
  Align, 982
  AlignButton, 984
  Alignment, 984
  AlignToBand, 985
  Allocation, 986
  AllocBy, 986
  AllowAllUp, 986
  AllowDelete, 987
  AllowGrayed, 987
  AllowInPlace, 987
  AllowInsert, 988
  AllowPaning, 988
  AllowResize, 989
  AllowSinglePoint, 989
  AllowZoom, 989
  ArrowKeys, 989
  AsBoolean, 990
  AsDateTime, 990
  AsFloat, 990
  AsInteger, 991
  Associate, 991
  AsString, 991
  AsText, 992
  AutoActivate, 992
  AutoCalcFields, 993
  AutoDisplay, 993
  AutoEdit, 994
  AutoEnable, 994
  AutoMerge, 994
  AutoOpen, 995
  AutoPopup, 995
  AutoRewind, 995
  AutoScroll, 996
  AutoSelect, 996
  AutoSize, 996
  AxisVisible, 997
  BackColor, 997
  BackgroundColor, 997
  BackImage, 998
  BevelInner, 998
  BevelOuter, 999
  BevelWidth, 999
  Bitmap, 1000
  BkColor, 1000
  BlendColor, 1000
  BlobSize, 1000
  BOF, 1001
  BorderIcons, 1001
  BorderStyle, 1001
  BorderWidth, 1002
  BottomAxis, 1003
  BoundsRect, 1003
  Break, 1004
  Brush, 1004
  ButtonAutoSize, 1005
  ButtonHeight, 1005
  ButtonSpacing, 1005
  ButtonWidth, 1005
  BytesRcvd, 1006
  BytesSent, 1006
  BytesTotal, 1006
  CachedSize, 1006
  CachedUpdates, 1006
  Calculated, 1007
  Cancel, 1007
  Canceled, 1007
  CanModify, 1008
  Canvas, 1008
  Capabilities, 1009
  Capacity, 1009
  Caption, 1010
  CaptionColor, 1010
  CaptionFont, 1010
  Category, 1011
  Cells, 1011

Center, 1011
ChangedCount, 1012
ChangedTableName, 1012
CharCase, 1012
Checked, 1013
ClientHandle, 1014
ClientHeight, 1014
ClientOrigin, 1014
ClientRect, 1015
ClientWidth, 1015
ClipRect, 1016
Col, 1016
ColCount, 1016
Collate, 1016
Color, 1017
ColorEachPoint, 1019
ColoredButtons, 1019
Cols, 1019
ColumnMarginMM, 1020
Columns, 1020
ColWidths, 1020
Command, 1021
CommandText, 1021
CommandTimeOut, 1021
CommandType, 1022
CommonAVI, 1022
ComponentCount, 1023
ComponentIndex, 1023
Components, 1024
ConfirmDelete, 1024
Connected, 1024
Connection, 1025
ConnectionString, 1025
ConnectMode, 1025
Context, 1026
ControlCount, 1026
Controls, 1027
ControlType, 1027
ConvertDlgHelp, 1028
Copies, 1028
Count, 1028
Ctl3D, 1029
Currency, 1029
CurrentMemory, 1030
CurrentPage, 1030
CurrentSum, 1030
Cursor, 1030
Cursors, 1031
CursorXPos, 1032
CursorYPos, 1032
CustomColors, 1033
Data, 1033
Database, 1033
DatabaseCount, 1033
DatabaseName, 1034
Databases, 1034
DataField, 1034
DataSet, 1035
DataSetCount, 1035
DataSets, 1035
DataSize, 1036
DataSource, 1036
DataType, 1037
DBFileName, 1037
DBHandle, 1037
DBLocale, 1037
DBSQLDialect, 1038
DDEConv, 1038
DDEItem, 1038
DDEService, 1039
DDETopic, 1039
DecisionCube, 1039
DecisionSource, 1039
Default, 1040
DefaultColWidth, 1040
DefaultDatabase, 1041
DefaultDrawing, 1041
DefaultExt, 1041
DefaultIndex, 1042
DefaultRowHeight, 1042
DefaultTransaction, 1042
DeleteSQL, 1042
Destination, 1043
Device, 1043
DeviceID, 1044
DeviceType, 1044
DimensionMap, 1044
Directory, 1045
DirLabel, 1045
DirList, 1045
DisableIfNoHandler, 1046
Display, 1046
DisplayFormat, 1047
DisplayLabel, 1047
DisplayName, 1047
DisplayPrintDialog, 1047
DisplayRect, 1048
DisplayText, 1048
DisplayValue, 1049
DisplayWidth, 1049
DitherBackground, 1049
DocBackColor, 1050
DocForeColor, 1050
DockClientCount, 1050
DockClients, 1050
DockRect, 1050
DockSite, 1051
DocLinkColor, 1051
Down, 1051
DragCursor, 1052
Dragging, 1052
DragMode, 1052
DrawingStyle, 1053
Drive, 1053
DriverName, 1054
DropDownAlign, 1054
DropDownCount, 1055
DropDownWidth, 1055
DropTarget, 1055
EditFormat, 1056
EditMask, 165, 1056
EditMaskPtr, 1057
EditMode, 1058
EditorMode, 1058
EditText, 1058
Empty, 1059
Enabled, 1059
EnabledButtons, 1059
EnableOpenBtn, 1060
EnablePrintBtn, 1060
EnableSaveBtn, 1060
EndMargin, 1060
EndPos, 1061
EOF, 1061
Error, 1061
ErrorCode, 1062
ErrorCount, 1062
ErrorMessage, 1062
Errors, 1063
Events, 1063
Exclusive, 1063
ExeName, 1063
Exists, 1064
Expanded, 1064
ExtendSelect, 1064
FieldClass, 1064
FieldCount, 1065
FieldDefs, 1065
FieldList, 1065
FieldName, 1065
FieldNo, 1066
Fields, 1066
FieldValues, 1066
FileEdit, 1067
FileEditStyle, 1067
FileExtension, 1067
FileList, 1068
FileName, 1068
FileType, 1069
Filter, 1070
Filtered, 1070
FilterIndex, 1070
FindText, 1071
FirstIndex, 1071
FirstPage, 1071
FixedColor, 1072
FixedCols, 1072
FixedRows, 1072
FocusControl, 1073
Focused, 1073
Font, 1073
Fonts, 1074
FooterBand, 1074
ForceNewPage, 1075
FormatChars, 1075
Formats, 1075
FormCount, 1076
Forms, 1076
FormStyle, 1076
Frame, 1077
FrameCount, 1077
FrameHeight, 1077
Frames, 1078
FrameWidth, 1077
FreeOnTerminated, 1078
Frequency, 1078
FromPage, 1078
FullPath, 1079
Glyph, 211, 1079
Graphic, 1080
GridHeight, 1080
GridLineWidth, 1080
GridWidth, 1081
GroupIndex, 1081
Handle, 1081
HandleIsShared, 1084
HasItems, 1084
HasProvider, 1085
Head1Font, 1085
Head2Font, 1085
Head3Font, 1085
Head4Font, 1086
Head5Font, 1086
Head6Font, 1086
HeaderBand, 1086
Height, 1087
HelpContext, 302, 1087
HelpFile, 1088
HelpKeyWord, 1088
HelpType, 1088
HideScrollBars, 1089
HideSelection, 1089
Hint, 1089
HintColor, 1090
HintPause, 1090
Hints, 206, 1091
HistoryList, 1091
HorizAxis, 1091
HorzScrollBar, 1092
Host, 1092
HotKey, 1092
Icon, 1093
IdleTimer, 1093
ImageIndex, 1093
Images, 1094
ImageType, 1094
Increment, 1094
Indent, 1095
Index, 1095
IndexDefs, 1095
IndexFieldCount, 1096
IndexFieldNames, 1096
IndexFields, 1096
IndexName, 1096
InitialDir, 1097
InPlaceActive, 1097
InsertSQL, 1097
IntegralHeight, 1098
Interval, 1098
Intransaction, 1098
InvalidKeys, 1099
IsIndexField, 1099
IsMasked, 1099
IsNull, 1100
IsReadOnly, 1100
IsSQLBased, 1100
IsVisible, 1101
ItemCount, 1101
ItemHeight, 1101
ItemIndex, 1102
Items, 1102
ItemSeparator, 1103
KeepConection, 1103
KeepConnections, 1103
KeyExclusive, 1104
KeyField, 1104
KeyFieldCount, 1104
KeyPreview, 1104
KeyValue, 1105
KeyViolCount, 1105
KeyViolTableName, 1105
Kind, 1106
LargeChange, 1107
LargeImages, 1107

LastPage, 1107
Layout, 1108
Left, 1108
LeftAxis, 1109
LeftCol, 1109
LeftMarginInches, 1109
LeftMarginMM, 1109
Length, 1110
Level, 1110
Lines, 1110
LinkBand, 1111
List, 1111
ListField, 1111
ListSource, 1112
Local, 1112
Locale, 1112
LocalIP, 1112
Locked, 1113
LoginPrompt, 1113
LookupDisplay, 1113
LookupField, 1114
LookupSource, 1114
MainForm, 1114
Mappings, 1115
Margin, 1115
MarginBotton, 1116
MarginLeft, 1116
MarginRight, 1116
MarginTop, 1116
Mask, 1117
Masked, 1117
Master, 1117
MasterFields, 1118
MasterSource, 1118
Max, 1118
MaxCells, 1118
MaxFontSize, 1119
MaxLength, 1119
MaxPage, 1119
MaxSummaries, 1120
MaxValue, 1120
MDIChildCount, 1120
MDIChildren, 1121
Menu, 1121
Message, 302, 1121
Metafile, 1122
Min, 1122
MinFontSize, 1122
MinPage, 1123
MinValue, 1123
ModalResult, 1123
Mode, 1124
Modified, 1126
Modifiers, 1126
ModifySQL, 1127
Monochrome, 1127
MovedCount, 1127
Multiline, 1128
MultiSelect, 1128
Name, 1128
NameList, 1129
NetFileDir, 1129
Notify, 1129
NotifyValue, 1130
NumGlyphs, 1130
ObjClass, 1131
ObjDoc, 1131
Objects, 1131
ObjItem, 1131
OEMConvert, 1132
Open, 1132
Operation, 1132
Options, 1133
Orientation, 1139
OriginalException, 1140
OutlineStyle, 1141
Overload, 1141
Owner, 1142
PageCount, 1142
PageHeight, 1142
PageIndex, 1143
PageNumber, 1143
Pages, 1144
PageSize, 1144
PageWidth, 1144
Palette, 1145
PanelBorder, 1145
PanelCount, 1145
PanelHeight, 1146
PanelIndex, 1146
Panels, 1146
PanelWidth, 1146
Paragraph, 1147
ParamBindMode, 1147
ParamCheck, 1147
ParamCount, 1148
Parameters, 1148
Params, 1148
Parent, 1149
ParentChart, 1149
ParentColor, 1149
ParentCtl3D, 1150
ParentFont, 1150
ParentShowHint, 1151
PasswordChar, 1151
Pen, 1152
PenPos, 1152
Picture, 1152
PictureClosed, 1152
PictureLeaf, 1153
PictureMinus, 1153
PictureOpen, 1154
PicturePlus, 1154
Pixels, 1154
PixelsPerInch, 1155
PlainText, 1155
PopupComponent, 1155
PopupMenu, 1156
Port, 1156
Position, 1156
PostMessage, 1157
Precision, 1158
Prepared, 1158
PreviousError, 1158
PrintBefore, 1158
PrinterIndex, 1159
PrinterOK, 1159
Printers, 1159
Printing, 1159
PrintMask, 1160
PrintRange, 1160
PrintScale, 1160
PrintToFile, 1161
PrivateDir, 1161
ProblemCount, 1162
ProblemTableName, 1162
ProcedureName, 1162
ProjectFile, 1163
Provider, 1163
ProviderName, 1163
QDelete, 1163
QInsert, 1164
QModify, 1164
QRefresh, 1164
QSelect, 1164
Query, 1165
Queued, 1165
RDSConnection, 1166
ReadOnly, 1165
RecNo, 1166
RecordCount, 1166
RecordSize, 1196
RefreshSQL, 1167
Registered, 1167
RemoteServer, 1167
Repetitions, 1167
ReplaceText, 1168
ReportTitle, 1168
ReportType, 1168
RequestURL, 1169
Required, 1169
ResetGroup, 1169
ResHandle, 1169
ResId, 1170
ResName, 1170
RestartData, 1170
RightAxis, 1170
Row, 1171
RowCount, 1171
RowHeights, 1172
Rows, 1172
RowsAffected, 1172
Ruler, 1173
Scaled, 1173
ScrollBars, 1173
ScrollPos, 1174
Sections, 1174
SectionWidth, 1175
SelCount, 1175
Selected, 1176
SelectedColor, 1176
SelectedField, 1176
SelectedIndex, 1176
Selection, 1177
SelectSQL, 1178
SelEnd, 1178
SelLength, 1178
SelStart, 1179
SelText, 1179
Series, 1179
SeriesColor, 1180
ServerConv, 1180
ServiceApplication, 1180
Shape, 1181
Sharable, 1182
ShareImages, 1182
ShareInLegend, 1185
ShortCurt, 1182
ShowAccelChar, 1183
ShowButtons, 1183
ShowFocus, 1184
ShowHint, 1184
Showing, 1184
ShowLines, 1185
ShowProgress, 1185
ShowRoot, 1186
SimplePanel, 1186
SimpleText, 1186
Size, 1187
SmallChange, 1187
SmallImages, 1187
Sorted, 1187
SortType, 1188
Source, 1188
Spacing, 1189
SparseCols, 1189
SparseRows, 1190
SQL, 1190
SQLCompatible, 1190
SQLConnection, 1191
SQLDialect, 1191
SQLObjectCount, 1191
SQLObjects, 1191
Start, 1192
StartFrame, 1192
StartMargin, 1192
StartPos, 1192
State, 1193
StateImages, 1194
Status, 1194
Step, 1195
StmtHandle, 1195
StopFrame, 1195
Storage, 1195
StoredDefs, 1196
StoredProcName, 1196
Stretch, 1196
Strings, 1197
Style, 1197
Suspended, 1200
TabHeight, 1200
TabIndex, 1201
Table, 1201
TableDirect, 1201
TableLevel, 1201
TableName, 1201
TableType, 1202
TabOrder, 1202
Tabs, 1203
TabsPerRow, 1203
TabStop, 1203
TabStops, 1204
TabWidth, 1204
Tag, 1204
Temporary , 1205
Terminated, 1205
Text, 1205
TextAlign, 1206
TextCase, 1207
ThreadID, 1207
TickMarks, 1207
TickStyle, 1208
TileMode, 1208
TimeFormat, 1209

Timers, 1209
Title, 1209
TitleBeforeHeader, 1210
TitleFont, 1210
Top, 1211
ToPage, 1211
TopAxis, 1211
TopIndex, 1212
TopItem, 1212
TopRow, 1212
TraceFlags, 1213
TrackLength, 1213
TrackPosition, 1213
Tracks, 1214
TransactionCount, 1214
Transactions, 1214
Transisolation, 1214
Transliterate, 1215
Transparent, 1215
Unidirecional, 1215
UnSelectedColor, 1216
UpdatedMode, 1216
UseCompression, 1217
Value, 1217
ValueChecked, 1218
Values, 1218
ValueUnChecked, 1219
VertScrollBar, 1220
ViewOrigin, 1220
ViewStyle, 1220
Visible, 1221
VisibleButtons, 1221
VisibleColCount, 1223
VisibleRowCount, 1223
VisibleTabs, 1224
Wait, 1224
WantReturns, 1224
WantTabs, 1225
Width, 1225
WindowMenu, 1225
WindowState, 1226
WordWrap, 1226
Wrap, 1227
WSAInfo, 1227
Zoom, 1227
Propriedades, definição de, 16
Provider, Propriedade, 1163
ProviderName, Propriedade, 1163
Push, Método, 1308

Q

QDelete, Propriedade, 1163
QInsert, Propriedade, 1164
QModify, Propriedade, 1164
QRefresh, Propriedade, 1164
QSelect, Propriedade, 1164
Query, Propriedade, 1165
QueryTableProducer Componente, 788
Queued, Propriedade, 1165
QueueEvents, Método, 1308
QuotedStr, função, 667

R

RAD, ferramentas, 4
RadToCicle, função, 826
RadToDeg, função, 827
RadToGrad, função, 827
RandG, função, 827
Rave Reports, 236
Rave Reports, ambiente de desenvolvimento, 237
RDSConnection, Propriedade, 1166
Read, Método, 1309
ReadBool, Método, 1309
ReadInteger, Método, 1309
ReadOnly, Propriedade, 1165
ReadSection, Método, 1309
ReadSectionValues, Método, 1309
ReadString, Método, 1310
Realign, Método, 1310
RecNo, Propriedade, 1166
RecordCount, Propriedade, 1166
RecordSize, Propriedade, 1196
Rectangle, Método, 1310
Refresh, Método, 1310
RefreshSQL, Propriedade, 1167
RegisterChanges, Método, 1311
Registered, Propriedade, 1167
RegisterEvents, Método, 1311
Relacionais, operadores, 56
Relatórios, criação de, 236
Release, Método, 1311
ReleaseHandle, Método, 1311
ReleasePalette, Método, 1311
RemoteServer, Propriedade, 1167
Remove, Método, 1312
RemoveAllPassword, Método, 1312
RemoveAllSeries, Método, 1312
RemoveComponent, Método, 1312
RemoveDatabase, Método, 1313
RemoveDatabases, Método, 1313
RemoveDir, função, 662
RemovePassword, Método, 1313
RemoveSeries, Método, 1313
RemoveTransaction, Método, 1314
RenameFile, Método, 662
Repaint, Método, 1314
Repetitions, Propriedade, 1167
ReplaceIcon, Método, 1314
ReplaceMasked, Método, 1314
ReplaceText, Propriedade, 1168
ReportTitle, Propriedade, 1168
ReportType, Propriedade, 1168
RequestData, Método, 1315
RequestURL, Propriedade, 1169
Required, Propriedade, 1169
Reset, Método, 1315
ResetGroup, Propriedade, 1169
ResHandle, Propriedade, 1169
ResId, Propriedade, 1170
ResName, Propriedade, 1170
ResourceLoad, Método, 1315
RestartData, Propriedade, 1170
Restore, Método, 1315
RestoreTopMosts, Método, 1316
Rewind, Método, 1316
RightAxis, Propriedade, 1170
RmDir, função, 662
Rollback, Método, 1317
RollBackRetaining, Método, 1317
RoundRect, Método, 1317
Row, Propriedade, 1171
RowCount, Propriedade, 1171
RowHeights, Propriedade, 1172
Rows, Propriedade, 1172
RowsAffected, Propriedade, 1172
Ruler, Propriedade, 1173
Run, Método, 1317
RvProject, componente, 236

S

Save, Método, 1317
SaveToBitmapFile, Método, 1318
SaveToFile, Método, 170, 1318
SaveToMetaFile, Método, 1318
SaveToMetaFileEnh, Método,1319
SaveToStream, Método, 1319
ScaleBy, Método,1319
Scaled, Propriedade, 1173
ScreenToClient, Método, 1319
ScrollBars, Propriedade, 1173
ScrollBy, Método, 1320
ScrollInView, Método, 1320
ScrollPos, Propriedade, 1174
Sections, Propriedade, 1174
SectionWidth, Propriedade, 1175
Seek, Método, 1321
SelCount, Propriedade, 1175
SelectAll, Método, 1321
Selected, Propriedade, 1176
SelectedColor, Propriedade, 1176
SelectedField, Propriedade, 1176
SelectedIndex, Propriedade, 1176
Selection, Propriedade, 1177
SelectNext, Método, 1321
SelectNextPage, Método, 1322
SelectSQL, Propriedade, 1178
SelEnd, Propriedade, 1178
Self, identificador, 296
SelLength, Propriedade, 1178
SelStart, Propriedade, 1179
SelText, Propriedade, 1179
SendToBack, Método, 1322
Series, Propriedade, 1179
SeriesColor, Propriedade, 1180
SeriesCount, Método, 1322
Server Login, caixa de diálogo, 131
ServerConv, Propriedade, 1180
ServiceApplication, Propriedade, 1180
SetAsHandle, Método, 1322
SetBounds, Método, 1323
SetComponent, Método, 1323
SetCurrentDir, função, 662
SetData, Método, 1323
SetFields, Método, 1323
SetFocus, Método, 1324
SetKey, Método, 1324
SetLength, função, 667
SetLink, Método, 1324
SetParams, Método, 1324
SetPrincipal, Método, 1324
SetPrinter, Método, 1325
SetRange, Método, 1325
SetRangeEnd, Método, 1325
SetRangeStart, Método, 1325
SetSelTextBuf, Método, 1326
SetString, função, 667
SetTabFocus, Método, 1326
SetText, Método, 1326
SetTextBuf, Método, 1326
SetUpdateState, Método, 1327
SetWindowExtEx, função, 632
Shape, componente, 343
Shape, Propriedade, 1181
Sharable, Propriedade, 1182
ShareImages, Propriedade, 1182
ShareInLegend, Propriedade, 1185
ShortCurt, Propriedade, 1182
Show, Método, 1327
ShowAccelChar, Propriedade, 1183
ShowButtons, Propriedade, 1183
ShowColumnHeaders, Propriedade, 1183
ShowCubeDialog, Método, 1327
ShowDragImage, Método, 1328

ShowException, Método, 1328
ShowFocus, Propriedade, 1184
ShowHint, Propriedade, 1184
Showing, Propriedade, 1184
ShowLines, Propriedade, 1185
ShowMessage, procedimento, 60
ShowModal, Método, 1328
ShowProgress, Propriedade, 1185
ShowRoot, Propriedade, 1186
SimplePanel, Propriedade, 1186
SimpleText, Propriedade, 1186
Sin, função, 827
SinCos, função, 827
SinH, função, 827
Size, Propriedade, 1187
SmallChange, Propriedade, 1187
Sorted, Propriedade, 1187
SortType, Propriedade, 1188
Source, Propriedade, 1188
Spacing, Propriedade, 1189
SparseCols, Propriedade, 1189
SparseRows, Propriedade, 1190
SQL, linguagem, 216
SQL, Propriedade, 1190
SQLCompatible, Propriedade, 1190
SQLConnection, Propriedade, 1191
SQLDialect, Propriedade, 1191
SQLObjectCount, Propriedade, 1191
SQLObjects, Propriedade, 1191
Start, Propriedade, 1192
StartFrame, Propriedade, 1192
StartMargin, Propriedade, 1192
StartPos, Propriedade, 1192
StartRecording, Método, 1329
StartTransaction, Método, 1329
State, Propriedade, 1193
StateImages, Propriedade, 1194
Status, Propriedade, 1194
StdDev, função, 827
Step, Método, 1329
Step, Propriedade, 1195
StmtHandle, Propriedade, 1195
Stop, Método, 1329
StopFrame, Propriedade, 1195
Storage, Propriedade, 1195
Stored Procedures, definição de, 586
StoredDefs, Propriedade, 1196
StoredProcName, Propriedade, 1196
Str, função, 668
StrAlloc, função, 672
StrBufSize, função, 672
StrCat, função, 672
StrComp, função, 673
StrCopy, função, 673
StrEnd, função, 673
Stretch, Propriedade, 1196
StretchDraw, Método, 1330
StringOfChar, função, 668
StringReplace, função, 668
Strings de auxílio, parametrização de, 738
Strings, manipulação de, 662
Strings, Propriedade, 1197
StrLCat, função, 674
StrLComp, função, 674
StrLCopy, função, 674
StrLen, função, 674
StrLIComp, função, 674
StrLower, função, 674
StrMove, função, 675
StrNew, função, 675
StrPCopy, função, 675
StrRScan, função, 675
StrScan, função, 675
StrToCurr, função, 668
StrToDate, função, 668
StrToFloat, função, 669
StrToInt, função, 669
StrToIntDef, função, 669
StrToTime, função, 669
StrUpper, função, 675
Style, Propriedade, 1197
SubPropriedades, 86
Sum, função, 828
SumAndSquares, função, 828
SumInt, função, 828
SumOfSquares, função, 828
Suspend, Método, 1330
Suspended, Propriedade, 1200
Synchronize, Método, 1330

T

Tabelas,
  criação de, 130, 463
  definindo campos de preenchimento obrigatório de, 141
  definindo chaves primárias de, 184, 438
  definindo índices de, 141
  definindo nomes de campos de, 136
  definindo tamanhos de campos de, 139
  definindo tipos de campos de, 138
TabHeight, Propriedade, 1200
TabIndex, Propriedade, 1201
Table, Propriedade, 1201
TableDirect, Propriedade, 1201
TableLevel, Propriedade, 1201
TableName, Propriedade, 1201
TableType, Propriedade, 1202
TabOrder, Propriedade, 1202
Tabs, Propriedade, 1203
TabsPerRow, Propriedade, 1203
TabStop, Propriedade, 1203
TabStops, Propriedade, 1204
TabWidth, Propriedade, 1204
TAction, Classe, 858
TActionList, Classe, 858
TADOCommand, Classe, 525, 859
  principais Métodos, 526
  principais Propriedades, 525
TADOConnection, Classe, 512, 859
  principais Eventos, 516
TADOConnection, Classe, principais Métodos, 514
TADOConnection, Classe, principais Propriedades, 512
TADODataSet, Classe, 527, 860
TADODataset, Classe, principais Métodos, 528
TADODataset, Classe, principais Propriedades, 527
TADOQuery, Classe, 530, 860
  principais Métodos, 530
  principais Propriedades, 530
TADOStoredProc, Classe, 861
TADOTable, Classe, 529, 862
  principais Métodos, 530
  principais Propriedades, 529
Tag, Propriedade, 1204
Tan, função, 828
Tanh, função, 828
TAnimate, Classe, 863
TApplication, Classe, 863
TApplicationEvents, Classe, 863
TAutoIncField, Classe, 864
TBaseReport, Classe, 864
TBatchMove, Classe, 865
TBCDField, Classe, 865
TBDEDataset, Classe, 416
TBevel, Classe, 866
TBitBtn, Classe, 866
TBitmap, Classe, 626, 866
TBlobField, Classe, 867
TBlobStream, Classe, 867
TBooleanField, Classe, 868
TBrush, Classe, 868
TButton, Classe, 868
TBytesField, Classe, 869
TCanvas, Classe, 335
TChart, Classe, 870
TChartSeries, Classe, 871
TCheckBox, Classe, 871
TClientDataSet, Classe, 872
TClientSocket, Classe, 873
TClipboard, Classe, 873
TColorDialog, Classe, 873
TComboBox, Classe, 874
TComponent, Classe, 874
TControlScrollBar, Classe, 875
TCorbaConnection, Classe, 875
TCorbaDatamodule, Classe, 729
TCurrencyField, Classe, 875
TCustomADODataSet, Classe, 518, 876
  principais Eventos, 524
  principais Métodos, 521
TCustomConnection, Classe, 411
TCustomSQLDataset, Classe, 543
TDataBase, Classe, 419, 876
TDataSet, Classe, 395
TDataSetPageProducer, Classe, 877
TDataSetProvider, Classe, 877
TDataSetTableProducer, Classe, 877
TDataSource, Classe, 878
TDateField, Classe, 878
TDateTime, tipo, 742
TDateTimeField, Classe, 878
TDBChart, Classe, 879
TDBCheckBox, Classe, 879
TDBComboBox, Classe, 880
TDBCtrlGrid, Classe, 880
TDBDataset, Classe, 418
TDBEdit, Classe, 881
TDBGrid, Classe, 881
TDBImage, Classe, 882
TDBListBox, Classe, 882
TDBLookupComboBox, Classe, 883
TDBLookupListBox, Classe, 883
TDBMemo, Classe, 884
TDBMemoBuf, Classe, 884
TDBNavigator, Classe, 885
TDBRadioGroup, Classe, 885
TDBRichEdit, Classe, 886
TDBText, Classe, 886
TDCOMConnection, Classe, 887
TDDEClientConv, Classe, 887
TDDEClientItem, Classe, 887
TDDEServerConv, Classe, 888
TDDEServerItem, Classe, 888
TDecisionCube, Classe, 888
TDecisionGraph, Classe, 888
TDecisionGrid, Classe, 889
TDecisionPivot, Classe, 890
TDecisionQuery, Classe, 890
TDecisionSource, Classe, 890
TDirectoryListBox, Classe, 891
TDrawGrid, Classe, 891
TDriveComboBox, Classe, 892
TEdit, Classe, 892
Template de formulários, definição, 210
Temporary, Propriedade, 1205
Terminate, Método, 1330

Terminated, Propriedade, 1205
TestConnected, Método, 1331
Testes condicionais, 56
Text, Propriedade, 1205
TextAlign, Propriedade, 1206
TextCase, Propriedade, 1207
TextHeight, Método, 1331
TextOut, Método, 1331
TextRect, Método, 1331
TextWidth, Método, 1331
TField, Classe, 893
TFieldDef, Classe, 893
TFileListBox, Classe, 893
TFilterComboBox, Classe, 894
TFindDialog, Classe, 894
TFloatField, Classe, 895
TFont, Classe, 895
TFontDialog, Classe, 895
TForm, Classe, 68, 896
TGraphic, Classe, 896
TGraphicsField, Classe, 897
TGroupBox, Classe, 897
THeader, Classe, 897
THeaderControl, Classe, 898
THotKey, Classe, 898
ThreadID, Propriedade, 1207
Threads, conceito de, 706
THTTPFile, Classe, 907
TIBClientDataSet, Classe, 899
TIBCustomDataSet, Classe, 558
  principais Eventos, 561
  principais Métodos, 560
  principais Propriedades, 559
  principais Eventos, 554
  principais Métodos, 553
  principais Propriedades, 552
TIBDatabaseInfo, Classe, 900
TIBDataSet, Classe, 561, 901
  principais Métodos, 565
  principais Propriedades, 563
TIBEvents, Classe, 901
TIBQuery, Classe, 567, 902
  principais Métodos, 568
  principais Propriedades, 567
TIBSQL, Classe, 902
TIBSQLMonitor, Classe, 903
TIBStoredProc, Classe,
  principais Eventos, 602
  principais Métodos, 601
  principais Propriedades, 601
TIBStoredProc, Classe, 903
TIBTable, Classe, 563, 904
  principais Métodos, 565
  principais Propriedades, 563
TIBTransaction, Classe, 558, 905
  principais Eventos, 558
  principais Métodos, 556
  principais Propriedades, 555
TIBUpdateSQL, Classe, 568, 905
  principais Métodos, 569
  principais Propriedades, 569
TickMarks, Propriedade, 1207
TickStyle, Propriedade, 1208
TIcon, Classe, 905
Tile, Método, 1332
TileMode, Propriedade, 1208
TImage, Classe, 906
TImageList, Classe, 906
TimeFormat, Propriedade, 1209
Timers, Propriedade, 1209
TimeToStr, função, 669
TIniFile, Classe, 907
TIntegerField, Classe, 907
Tipos de dados enumerados, 47
Tipos ordinais, 52
Tipos, conversão de, 294
Title, Propriedade, 1209
TitleBeforeHeader, Propriedade, 1210
TitleFont, Propriedade, 1210
Títulos, barra de, 11
TIWAppForm, Classe, 908
TIWApplet, Classe, 908
TIWApplication, Classe, 908
TIWButton, Classe, 909
TIWCheckBox, Classe, 909
TIWComboBox, Classe, 910
TIWControl, Classe, 910
TIWDBCheckBox, Classe, 910
TIWDBComboBox, Classe, 911
TIWDBEdit, Classe, 911
TIWDBFile, Classe, 911
TIWDBGrid, Classe, 912
TIWDBImage, Classe, 912
TIWDBListBox, Classe, 913
TIWDBLookupComboBox, Classe, 913
TIWDBLookupListBox, Classe, 913
TIWDBMemo, Classe, 914
TIWDBNavigator, Classe, 914
TIWDBText, Classe, 915
TIWEdit, Classe, 915
TIWFile, Classe, 915
TIWForm, Classe, 916
TIWGrid, Classe, 916
TIWImage, Classe, 917
TIWImageFile, Classe, 917
TIWLabel, Classe, 917
TIWLink, Classe, 918
TIWList, Classe, 918
TIWListBox, Classe, 918
TIWMemo, Classe, 919
TIWRectangle, Classe, 919
TIWRegion, Classe, 920
TIWTimer, Classe, 920
TIWTreeView, Classe, 920
TIWURL, Classe, 921
TLabel, Classe, 921
TList, Classe, 305, 921
TListBox, Classe, 922
TListView, Classe, 922
TMaskEdit, Classe, 923
TMediaPlayer, Classe, 924
TMemo, Classe, 924
TMemoBuf, Classe, 925
TMemoField, Classe, 925
TMenuItem, Classe, 925
TMetafile, Classe, 926
TMIDASConnection, Classe, 926
TMIDASPageProducer, Classe, 926
TMTSDatamodule, Classe, 729
TNMDayTime, Classe, 927
TNMEcho, Classe, 927
TNMFinger, Classe, 927
TNMFTP, Classe, 928
TNMHTTP, Classe, 928
TNMMSG, Classe, 928
TNMMSGServ, Classe, 929
TNMNNTP, Classe, 929
TNMPOP3, Classe, 929
TNMSMTP, Classe, 930
TNMSTRM, Classe, 930
TNMSTRMServ, Classe, 931
TNMTime, Classe, 931
TNMUDP, Classe, 931
TNMURL, Classe, 932
TNMUUProcessor, Classe, 930
Tnotebook, Classe, 932
TObject, Classe, Classe, Classe, 287
TODO Lists,
  adicionando um item a uma, 77, 80
  conceito, 77
  copiando a relação de itens de uma, 81
  editando um item de uma, 78
  excluindo um item de uma, 79
  filtrando itens de uma, 81
TOLEContainer, Classe, 933
Top, Propriedade, 1211
ToPage, Propriedade, 1211
TopAxis, Propriedade, 1211
TOpenDialog, Classe, 933
TOpenPictureDialog, Classe, 933
TopIndex, Propriedade, 1212
TopItem, Propriedade, 1212
TopRow, Propriedade, 1212
TOutline, Classe, 934
TOutlineNode, Classe, 934
TPageControl, Classe, 935
TPageProducer, Classe, 935
TPaintBox, Classe, 935
TPanel, Classe, 936
TPen, Classe, 341, 936
TPicture, Classe, 937
TPopupMenu, Classe, 937
TPrintDialog, Classe, 937
TPrinter, Classe, 637, 938
TPrinterSetupDialog, Classe, 938
TProgressBar, Classe, 939
TProvider, Classe, 939
TQRDBText, Classe, 939
TQRLabel, Classe, 940
TQRMemo, Classe, 940
TQRPreview, Classe, 940
TQRPrinter, Classe, 941
TQRShape, Classe, 941
TQRSysData, Classe, 941
TQuery, Classe, 434, 942
TQueryTableProducer, Classe, 942
TQueue, Classe, 942
TQuickRep, Classe, 943
TraceFlags, Propriedade, 1213
TrackLength, Propriedade, 1213
TrackPosition, Propriedade, 1213
Tracks, Propriedade, 1214
TRadioButton, Classe, 943
TRadioGroup, Classe, 944
Tradução, ambiente integrado de, 758
Transações, conceito de, 583
TransactionCount, Propriedade, 1214
Transactions, Propriedade, 1214
Transisolation, Propriedade, 1214
Translation Repository, 766
Transliterate, Propriedade, 1215
Transparent, Propriedade, 1215
TRDSConnection, Classe, 517, 944
  principais Métodos, 517
  principais Propriedades, 517
TRegistry, Classe, 945
TRegistryIniFile, Classe, 945
TRemoteServer, Classe, 945
TReplaceDialog, Classe, 946
TRichEdit, Classe, 946
Triggers, definição de, 585
Triggers, tipos de, 585
Trim, função, 669
TrimLeft, função, 669
TrimRight, função, 670
TRpBarsBase, Classe, 946
TRpBaseComponent, Classe, 947
TRpComponent, Classe, 947
TRpRender, Classe, 948
TRpRenderCanvas, Classe, 948
TRpRenderStream, Classe, 948
Truncate, Método, 1332
TRvCustomConnection, Classe, 949
TRvDatsetConnection, Classe, 949
TRvNDRWriter, Classe, 949
TRvProject, Classe, 950
TRvQueryConnection, Classe, 954
TRvRenderHTML, Classe, 951
TRvRenderPDF, Classe, 951
TRvRenderPreview, Classe, 951
TRvRenderPrinter, Classe, 952
TRvRenderRTF, Classe, 953
TRvRenderText, Classe, 954

TRvSystem, Classe, 954
TRvTableConnection, Classe, 955
TSaveDialog, Classe, 955
TSavePictureDialog, Classe, 955
TScreen, Classe, 956
TScrollBar, Classe, 956
TScrollBox, Classe, 957
TServerSocket, Classe, 957
TSession, Classe, 422, 958
TShape, Classe, 958
TSimpleDataset, Classe, 548
  principais Métodos, 549
  principais Propriedades, 548
TSimpleObjectBroker, Classe, 959
TSmallIntField, Classe, 960
TSocketConnecttion, Classe, 960
TSpeedButton, Classe, 960
TSQLConnection, Classe, 540, 961
  principais Eventos, 543
  principais Métodos, 542
  principais Propriedades, 540
TSQLDataSet, Classe, 544, 961
TSQLDataSet Classe, principais Propriedades, 544
TSQLMonitor, Classe, 962
TSQLQuery Classe, 547, 962
  principais Métodos, 547
  principais Propriedades, 547
TSQLStoredProc, Classe, 605, 963
  principais Eventos, 606
  principais Métodos, 606
  principais Propriedades, 605
TSQLTable Classe, 546, 964
  principais Métodos, 547
  principais Propriedades, 546
TSQLTable, componente, 154
TStack, Classe, 964
TStatusBar, Classe, 965
TStoredProc, Classe, 595, 965
  principais Eventos, 597
  principais Métodos, 596
  principais Propriedades, 595
TStringField, Classe, 966
TStringGrid, Classe, 966
TStringList, Classe, 966
TStrings, Classe, 967
TTabbedNotebook, Classe, 967
TTabControl, Classe, 968
TTable, Classe, 428, 968
TTabSet, Classe, 969
TTabSheet, 969
TThread, Classe, 708, 970
TTimeField, Classe, 970
TTimer, Classe, 970
TToolBar, Classe, 971
TTrackBar, Classe, 971
TTreeView, Classe, 972
TUpdateSQL, Classe, 437, 500, 972
TUpDown, Classe, 973
TVarBytesField, Classe, 973
TWebDispatcher, Classe, 974
TWordField, Classe, 974
TXMLBroker, Classe, 974

U

Unidade de código, 40
  estrutura de uma, 40
  seções de, 41
Unidirecional, Propriedade, 1215
UnlockTable, Método, 1332
UnMerge, Método, 1332
UnPrepare, Método, 1333
UnRegisterChanges, Método, 1333
UnRegisterEvents, Método, 1333
UnSelectedColor, Propriedade, 1216
Update, Método, 1333
UpdateCursrPos, Método, 1334
UpdatedMode, Propriedade, 1216
UpdateRecord, Método, 1334
Uppercase, função, 670
UseCompression, Propriedade, 1217

V

Val, função, 670
ValidateEdit, Método, 1334
Value List Editor, caixa de diálogo, 1
Value, Propriedade, 1217
ValueChecked, Propriedade, 1218
Values, Propriedade, 1218
ValueUnChecked, Propriedade, 1219
Variance, função, 828
Variáveis,
  atribuição de valor a, 43
  booleanas, 44
  compostas, 49
  conceito de, 43
  declaração de, 42
  escopo, 52
  globais, 53
  inteiras, 43
  locais, 52
  para manipulação de arquivos, 45
  para manipulação de caracteres, 44
  reais, 44
  tipo genérico de, 46
  tipos de, 43
VertScrollBar, Propriedade, 1220
Vetores, 49
ViewOrigin, Propriedade, 1220
ViewStyle, Propriedade, 1220
Visible, Propriedade, 1221
VisibleButtons, Propriedade, 1221
VisibleColCount, Propriedade, 1223
VisibleRowCount, Propriedade, 1223
VisibleTabs, Propriedade, 1224
Visões, 582

W

Wait, Propriedade, 1224
WantReturns, Propriedade, 1224
WantTabs, Propriedade, 1225
WebModule, criação de, 772
WebSnap, Tecnologia, 791
Width, Propriedade, 1225
WindowMenu, Propriedade, 1225
WindowState, Propriedade, 1226
WordWrap, Propriedade, 1226
Wrap, Propriedade, 1227
WrapText, função, 670
Write, Método, 1334
WriteBCDData, Método, 1335
WriteBlobData, Método, 1335
WriteBool, Método, 1335
WriteBoolData, Método, 1335
WriteCurrData, Método, 1335
WriteDateTime, Método, 1336
WriteFloatData, Método, 1336
WriteIntData, Método, 1336
WriteInteger, Método, 1366
WriteNullData, Método, 1336
WriteStrData, Método, 1337
WriteString, Método, 1337
WSAInfo, Propriedade, 1227

Z

Zoom, Propriedade, 1227
ZoomToFit, Método, 1337
ZoomToWidth, Método, 1337