# Table of Contents

## Part I  What's New
1

## Part II  General Information
7

1 Overview .................................................................................................................. 7
2 Features .................................................................................................................. 10
3 Requirements ........................................................................................................ 14
4 Compatibility ......................................................................................................... 14
5 Using Several DAC Products in One IDE ............................................................ 19
6 Component List ..................................................................................................... 19
7 Hierarchy Chart ..................................................................................................... 21
8 Editions .................................................................................................................. 23
9 Licensing ................................................................................................................ 25
10 Getting Support .................................................................................................... 29
11 FAQ ...................................................................................................................... 30

## Part III  Getting Started
37

1 Installation ............................................................................................................. 42
2 Connecting to PostgreSQL .................................................................................... 44
3 Deleting Data From Tables .................................................................................. 48
4 Creating Database Objects ................................................................................... 52
5 Inserting Data Into Tables ..................................................................................... 56
6 Retrieving Data From Tables ................................................................................ 62
7 Modifying Data in Tables ...................................................................................... 64
8 Using Stored Procedures ....................................................................................... 67
9 Using Stored Functions with Result Sets ............................................................. 73
10 Demo Projects ...................................................................................................... 76
11 Deployment .......................................................................................................... 81

## Part IV  Using PgDAC
82

1 Updating Data with PgDAC Dataset Components .............................................. 83
2 Master/Detail Relationships .................................................................................. 84
3 Automatic Key Field Value Generation ............................................................... 86
4 Data Type Mapping ............................................................................................... 87
5 Data Encryption ..................................................................................................... 93
6 Working in an Unstable Network ......................................................................... 95
7 Secure Connections ............................................................................................... 96
    Connecting via SSL ............................................................................................. 97
    Connecting via SSH ............................................................................................. 102
## Part V Reference

### CRAccess

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRCursor Class</td>
<td>147</td>
</tr>
<tr>
<td>TCRTransactionAction Enumeration</td>
<td>150</td>
</tr>
<tr>
<td>TCursorState Enumeration</td>
<td>151</td>
</tr>
</tbody>
</table>

### CRBatchMove

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMove Class</td>
<td>153</td>
</tr>
<tr>
<td>AbortOnKeyViol Property</td>
<td>157</td>
</tr>
<tr>
<td>AbortOnProblem Property</td>
<td>157</td>
</tr>
<tr>
<td>ChangedCount Property</td>
<td>158</td>
</tr>
<tr>
<td>CommitCount Property</td>
<td>158</td>
</tr>
<tr>
<td>Destination Property</td>
<td>159</td>
</tr>
<tr>
<td>FieldMappingMode Property</td>
<td>159</td>
</tr>
<tr>
<td>KeyViolCount Property</td>
<td>160</td>
</tr>
<tr>
<td>Mappings Property</td>
<td>160</td>
</tr>
<tr>
<td>Mode Property</td>
<td>161</td>
</tr>
<tr>
<td>MovedCount Property</td>
<td>161</td>
</tr>
<tr>
<td>ProblemCount Property</td>
<td>162</td>
</tr>
<tr>
<td>RecordCount Property</td>
<td>162</td>
</tr>
<tr>
<td>Source Property</td>
<td>163</td>
</tr>
</tbody>
</table>

---

Network Tunneling | 109
8 Disconnected Mode | 115
9 Batch Operations | 116
10 Increasing Performance | 120
11 Macros | 122
12 DataSet Manager | 123
13 TPgLoader Component | 129
14 Large Objects | 130
15 REFCURSOR Data Type | 131
16 National and Unicode Characters | 133
17 Connection Pooling | 134
18 DBMonitor | 136
19 Writing GUI Applications with PgDAC | 136
20 Compatibility with Previous Versions | 136
21 64-bit Development with Embarcadero RAD Studio XE2 | 138
22 Database Specific Aspects of 64-bit Development | 143

© 2021 Devart
<table>
<thead>
<tr>
<th>Contents</th>
<th>III</th>
</tr>
</thead>
</table>

3 **CREncryption** ................................................................. 168

**Classes** ................................................................................. 169

TCREncryptor Class .................................................................. 169

**Members** ................................................................................ 169

Properties .................................................................................. 170

- DataHeader Property .............................................................. 171
- EncryptionAlgorithm Property ................................................ 171
- HashAlgorithm Property ......................................................... 172
- InvalidHashAction Property ..................................................... 172
- Password Property ................................................................... 173

**Methods** ................................................................................ 173

- SetKey Method ......................................................................... 174

**Enumerations** ......................................................................... 175

- TCREncDataHeader Enumeration .............................................. 175
- TCREncryptionAlgorithm Enumeration ...................................... 176
- TCRClockAlgorithm Enumeration ............................................. 176
- TCRClockInvalidHashAction Enumeration ............................... 177

4 **CRGrid** .................................................................................. 177

**Classes** ..................................................................................... 178

TCRDBGrid Class ........................................................................ 178

**Members** .................................................................................. 178

Properties ...................................................................................... 181

- Columns Property ................................................................. 182
- Filtered Property ................................................................. 183
- LevelDelimiterChar Property .................................................. 183
- OnMemoClick Property .......................................................... 183
- OptionsEx Property ............................................................... 184
- OptionsMenu Property ........................................................... 185

**Methods** ................................................................................... 185

- ActivateFilterEdit Method ...................................................... 187
- ActivateSearchEdit Method ..................................................... 187
- AdjustColumns Method .......................................................... 187
- ApplyFilter Method ............................................................... 188
- CalcTitleLevel Method .......................................................... 188
- ClearFilters Method ............................................................... 189
- ClearSorting Method .............................................................. 189
- DataChanged Method ............................................................. 190
- GetGridSize Method .............................................................. 190
- GetTitleLevel Method ............................................................ 190

**Events** ....................................................................................... 191

- OnGetCellParams Event .......................................................... 191

5 **CRVio** ...................................................................................... 192

**Classes** ..................................................................................... 192

THttoptions Class ......................................................................... 193

**Members** .................................................................................. 193
IV PostgreSQL Data Access Components

Properties .............................................................. 194
  Enabled Property ............................................. 195
  Password Property ........................................... 195
  ProxyOptions Property ...................................... 195
  TrustServerCertificate Property .......................... 196
  Url Property ................................................... 196
  Username Property .......................................... 197

TProxyOptions Class .......................................... 197
  Members .......................................................... 198
  Properties ....................................................... 198
    Hostname Property ......................................... 199
    Password Property ......................................... 199
    Port Property ............................................... 200
    Username Property ........................................ 200

Enumerations ..................................................... 200
  TProxyVersion Enumeration .................................. 201

6 DAAlerter .......................................................... 201

Classes ................................................................. 202
  TDAlerter Class ............................................... 202
    Members ......................................................... 203
    Properties ...................................................... 204
      Active Property .......................................... 204
      AutoRegister Property .................................. 205
      Connection Property .................................... 205
    Methods ....................................................... 206
      SendEvent Method ....................................... 206
      Start Method ............................................. 207
      Stop Method .............................................. 207
    Events ......................................................... 208
      OnError Event ............................................ 208

Types ................................................................. 209
  TAlterErrorEvent Procedure Reference ................ 209

7 DADump ............................................................. 209

Classes ................................................................. 210
  TDADump Class ................................................ 211
    Members ....................................................... 211
    Properties ...................................................... 213
      Connection Property .................................... 214
      Debug Property .......................................... 214
      Options Property ........................................ 215
      SQL Property ............................................. 216
      TableName Property ...................................... 216
    Methods ....................................................... 217
      Backup Method .......................................... 218
      BackupQuery Method .................................... 218
      BackupToFile Method .................................... 219
      BackupToStream Method ................................ 220
      Restore Method ......................................... 220
      RestoreFromFile Method ................................ 221
      RestoreFromStream Method ............................. 222
    Events ......................................................... 222
      OnBackupProgress Event ............................... 223
      OnError Event ............................................ 224

© 2021 Devart
## 8 DALoader

### Classes
- **TDAColumn Class**
  - Members
  - Properties
    - FieldType Property
    - Name Property
- **TDAColumns Class**
  - Members
  - Properties
    - Items Property (Indexer)
- **TDALoader Class**
  - Members
  - Properties
    - Columns Property
    - Connection Property
    - TableName Property
  - Methods
    - CreateColumns Method
    - Load Method
    - LoadFromDataSet Method
    - PutColumnData Method
    - PutColumnData Method
    - PutColumnData Method
    - PutColumnData Method
  - Events
    - OnGetColumnData Event
    - OnProgress Event
    - OnPutData Event
- **TDALoaderOptions Class**
  - Members
  - Properties
    - UseBlankValues Property
- **Types**
  - TDAutoProgressEvent Procedure Reference
  - TDAutoProgressEvent Procedure Reference

## 9 DAScript

### Classes
- **TDAScript Class**
  - Members
  - Properties
    - Connection Property
Data Set Property ................................................................. 257
Debug Property ................................................................. 257
Delimiter Property .............................................................. 258
End Line Property .............................................................. 258
End Offset Property ............................................................. 259
End Position Property ......................................................... 259
Macros Property ................................................................. 259
SQL Property ...................................................................... 260
Start Line Property ............................................................. 260
Start Offset Property .......................................................... 261
Start Position Property ....................................................... 261
Statements Property ........................................................... 262

Methods .............................................................................. 263
Break Execute Method ......................................................... 264
Error Offset Method ............................................................ 264
Execute Method .................................................................. 265
Execute File Method ............................................................ 265
Execute Next Method ........................................................... 266
Execute Stream Method ....................................................... 266
Find Macro Method ............................................................. 267
Macro By Name Method ....................................................... 268

Events .................................................................................. 268
After Execute Event ............................................................ 269
Before Execute Event .......................................................... 270
OnError Event ..................................................................... 270

TDA Statement Class .......................................................... 271
Members ............................................................................. 271
Properties .......................................................................... 272
End Line Property .............................................................. 273
End Offset Property ............................................................. 274
End Position Property ......................................................... 274
Omit Property ..................................................................... 274
Params Property ................................................................. 275
Script Property ................................................................... 275
SQL Property ...................................................................... 276
Start Line Property ............................................................. 276
Start Offset Property .......................................................... 277
Start Position Property ....................................................... 277

Methods .............................................................................. 277
Execute Method .................................................................. 278

TDA Statements Class .......................................................... 278
Members ............................................................................. 279
Properties .......................................................................... 279
Items Property (Indexer) ....................................................... 280

Types .................................................................................. 280
TAfter Statement Execute Event Procedure Reference ...... 281
TBefore Statement Execute Event Procedure Reference ..... 281
TOnError Event Procedure Reference ................................. 282

Enumerations ........................................................................ 282
TError Action Enumeration .................................................. 283

10 DASQL Monitor ............................................................... 283

Classes ................................................................................. 284
TCustom DASQL Monitor Class ........................................... 285
Members ............................................................................. 285
Properties .......................................................................................................................................................... 286
Active Property .................................................................................................................................................. 287
DBMonitorOptions Property ......................................................................................................................... 287
Options Property ............................................................................................................................................... 287
TraceFlags Property ......................................................................................................................................... 288
Events ................................................................................................................................................................. 289
OnSQL Event ..................................................................................................................................................... 289
TDBMonitorOptions Class .................................................................................................................................. 290
Members ............................................................................................................................................................... 290
Properties ............................................................................................................................................................ 291
Host Property ...................................................................................................................................................... 291
Port Property ....................................................................................................................................................... 292
ReconnectTimeout Property .................................................................................................................................... 292
SendTimeout Property ........................................................................................................................................... 293
Types .................................................................................................................................................................. 293
TDATraceFlags Set............................................................................................................................................... 294
TMonitorOptions Set............................................................................................................................................ 294
TOnSQLEvent Procedure Reference................................................................................................................... 294
Enumerations ..................................................................................................................................................... 295
TDATraceFlag Enumeration ............................................................................................................................... 295
TMonitorOption Enumeration ............................................................................................................................ 296
11 DBAccess ...................................................................................................................................................... 297
Classes ................................................................................................................................................................. 300
EDAEError Class ................................................................................................................................................. 302
Members ............................................................................................................................................................... 303
Properties ............................................................................................................................................................ 303
Component Property .......................................................................................................................................... 304
ErrorCode Property ............................................................................................................................................ 304
TRCDataSource Class ......................................................................................................................................... 304
Members ............................................................................................................................................................... 305
Properties ............................................................................................................................................................ 306
CancelButton Property ........................................................................................................................................ 308
Caption Property ............................................................................................................................................... 308
ConnectButton Property ..................................................................................................................................... 309
DialogClass Property ......................................................................................................................................... 309
LabelSet Property ............................................................................................................................................... 310
PasswordLabel Property ...................................................................................................................................... 310
Retries Property .................................................................................................................................................. 311
SavePassword Property ...................................................................................................................................... 311
ServerLabel Property .......................................................................................................................................... 312
StoreLogInfo Property ........................................................................................................................................ 312
UsernameLabel Property ........................................................................................................................................ 313
Methods ................................................................................................................................................................. 313
Execute Method.................................................................................................................................................. 314
GetServerList Method ......................................................................................................................................... 314
TCustomDAConnection Class ................................................................................................................................ 315
Members ............................................................................................................................................................... 315
Properties ............................................................................................................................................................ 317
ConnectDialog Property ....................................................................................................................................... 319
ConnectString Property ....................................................................................................................................... 319
ConvertEOL Property .......................................................................................................................................... 321
InTransaction Property .......................................................................................................................................... 321
<table>
<thead>
<tr>
<th>PostgreSQL Data Access Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Properties</strong></td>
<td>322</td>
</tr>
<tr>
<td>LoginPrompt Property</td>
<td>322</td>
</tr>
<tr>
<td>Options Property</td>
<td>323</td>
</tr>
<tr>
<td>Password Property</td>
<td>324</td>
</tr>
<tr>
<td>Pooling Property</td>
<td>324</td>
</tr>
<tr>
<td>PoolingOptions Property</td>
<td>324</td>
</tr>
<tr>
<td>Server Property</td>
<td>325</td>
</tr>
<tr>
<td>Username Property</td>
<td>326</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td>327</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>327</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>329</td>
</tr>
<tr>
<td>ApplyUpdates Method</td>
<td>330</td>
</tr>
<tr>
<td>Commit Method</td>
<td>330</td>
</tr>
<tr>
<td>Connect Method</td>
<td>331</td>
</tr>
<tr>
<td>CreateSQL Method</td>
<td>332</td>
</tr>
<tr>
<td>Disconnect Method</td>
<td>332</td>
</tr>
<tr>
<td>ExecProc Method</td>
<td>333</td>
</tr>
<tr>
<td>ExecProcEx Method</td>
<td>334</td>
</tr>
<tr>
<td>ExecSQL Method</td>
<td>336</td>
</tr>
<tr>
<td>ExecSQLEx Method</td>
<td>337</td>
</tr>
<tr>
<td>GetDatabaseNames Method</td>
<td>338</td>
</tr>
<tr>
<td>GetKeyFieldNames Method</td>
<td>338</td>
</tr>
<tr>
<td>GetStoredProcNames Method</td>
<td>339</td>
</tr>
<tr>
<td>GetTableNames Method</td>
<td>340</td>
</tr>
<tr>
<td>MonitorMessage Method</td>
<td>341</td>
</tr>
<tr>
<td>Ping Method</td>
<td>342</td>
</tr>
<tr>
<td>RemoveFromPool Method</td>
<td>342</td>
</tr>
<tr>
<td>Rollback Method</td>
<td>343</td>
</tr>
<tr>
<td>StartTransaction Method</td>
<td>343</td>
</tr>
<tr>
<td><strong>Events</strong></td>
<td>344</td>
</tr>
<tr>
<td>OnConnectionLost Event</td>
<td>344</td>
</tr>
<tr>
<td>OnError Event</td>
<td>345</td>
</tr>
<tr>
<td><strong>Members</strong></td>
<td>345</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td>346</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td>347</td>
</tr>
<tr>
<td><strong>Events</strong></td>
<td>348</td>
</tr>
<tr>
<td><strong>Members</strong></td>
<td>349</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td>350</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td>351</td>
</tr>
<tr>
<td><strong>Events</strong></td>
<td>352</td>
</tr>
<tr>
<td><strong>Members</strong></td>
<td>353</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td>354</td>
</tr>
<tr>
<td>BaseSQL Property</td>
<td>358</td>
</tr>
<tr>
<td>Conditions Property</td>
<td>358</td>
</tr>
<tr>
<td>Connection Property</td>
<td>359</td>
</tr>
<tr>
<td>DataTypeMap Property</td>
<td>359</td>
</tr>
<tr>
<td>Debug Property</td>
<td>360</td>
</tr>
<tr>
<td>DetailFields Property</td>
<td>360</td>
</tr>
<tr>
<td>Disconnected Property</td>
<td>361</td>
</tr>
<tr>
<td>FetchRows Property</td>
<td>361</td>
</tr>
<tr>
<td>FilterSQL Property</td>
<td>362</td>
</tr>
<tr>
<td>FinalSQL Property</td>
<td>363</td>
</tr>
<tr>
<td>IsQuery Property</td>
<td>363</td>
</tr>
<tr>
<td>KeyFields Property</td>
<td>364</td>
</tr>
<tr>
<td>MacroCount Property</td>
<td>365</td>
</tr>
<tr>
<td>Macros Property</td>
<td>365</td>
</tr>
<tr>
<td>MasterFields Property</td>
<td>366</td>
</tr>
<tr>
<td>MasterSource Property</td>
<td>367</td>
</tr>
<tr>
<td>Options Property</td>
<td>368</td>
</tr>
<tr>
<td>ParamCheck Property</td>
<td>370</td>
</tr>
<tr>
<td>ParamCount Property</td>
<td>370</td>
</tr>
<tr>
<td>Params Property</td>
<td>371</td>
</tr>
</tbody>
</table>
PostgreSQL Data Access Components

TDACondition Class

Properties

Members

Methods

Enable Method

Value Property

Name Property

TCustomDAUpdateSQL Class

Properties

Members

Methods

BreakExec Method

Execute Method

Execute Method

Execute Method

Executing Method

FindMacro Method

FindParam Method

MacroByName Method

ParamByName Method

Prepare Method

UnPrepare Method

WaitExecuting Method

Events

AfterExecute Event

TCustomDAUpdateSQL Class

Properties

Members

Methods

Apply Method

ExecSQL Method

TDACondition Class

Properties

Members

Enabled Property

Name Property

Value Property

Methods

Disable Method

Enable Method
<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAConditions Class</td>
<td>444</td>
</tr>
<tr>
<td>Members</td>
<td>445</td>
</tr>
<tr>
<td>Properties</td>
<td>446</td>
</tr>
<tr>
<td>Condition Property (Indexer)</td>
<td>447</td>
</tr>
<tr>
<td>Enabled Property</td>
<td>447</td>
</tr>
<tr>
<td>Items Property (Indexer)</td>
<td>447</td>
</tr>
<tr>
<td>Text Property</td>
<td>448</td>
</tr>
<tr>
<td>WhereSQL Property</td>
<td>448</td>
</tr>
<tr>
<td>Methods</td>
<td>449</td>
</tr>
<tr>
<td>Add Method</td>
<td>450</td>
</tr>
<tr>
<td>Add Method</td>
<td>450</td>
</tr>
<tr>
<td>Add Method</td>
<td>451</td>
</tr>
<tr>
<td>Delete Method</td>
<td>451</td>
</tr>
<tr>
<td>Disable Method</td>
<td>452</td>
</tr>
<tr>
<td>Enable Method</td>
<td>452</td>
</tr>
<tr>
<td>Find Method</td>
<td>452</td>
</tr>
<tr>
<td>Get Method</td>
<td>454</td>
</tr>
<tr>
<td>IndexOf Method</td>
<td>453</td>
</tr>
<tr>
<td>Remove Method</td>
<td>454</td>
</tr>
<tr>
<td>TDAConnectionOptions Class</td>
<td>454</td>
</tr>
<tr>
<td>Members</td>
<td>454</td>
</tr>
<tr>
<td>Properties</td>
<td>455</td>
</tr>
<tr>
<td>Allow ImplicitConnect Property</td>
<td>456</td>
</tr>
<tr>
<td>DefaultSortType Property</td>
<td>457</td>
</tr>
<tr>
<td>DisconnectedMode Property</td>
<td>457</td>
</tr>
<tr>
<td>KeepDesignConnected Property</td>
<td>458</td>
</tr>
<tr>
<td>LocalFailover Property</td>
<td>459</td>
</tr>
<tr>
<td>TDAConnectionSSLOptions Class</td>
<td>459</td>
</tr>
<tr>
<td>Members</td>
<td>460</td>
</tr>
<tr>
<td>Properties</td>
<td>460</td>
</tr>
<tr>
<td>CA Cert Property</td>
<td>461</td>
</tr>
<tr>
<td>Cert Property</td>
<td>461</td>
</tr>
<tr>
<td>CipherList Property</td>
<td>461</td>
</tr>
<tr>
<td>Key Property</td>
<td>462</td>
</tr>
<tr>
<td>TDADataSetOptions Class</td>
<td>462</td>
</tr>
<tr>
<td>Members</td>
<td>463</td>
</tr>
<tr>
<td>Properties</td>
<td>465</td>
</tr>
<tr>
<td>AutoPrepare Property</td>
<td>468</td>
</tr>
<tr>
<td>CacheCalcFields Property</td>
<td>468</td>
</tr>
<tr>
<td>CompressBlobMode Property</td>
<td>468</td>
</tr>
<tr>
<td>DefaultValues Property</td>
<td>469</td>
</tr>
<tr>
<td>DetailDelay Property</td>
<td>469</td>
</tr>
<tr>
<td>FieldsOrigin Property</td>
<td>470</td>
</tr>
<tr>
<td>FlatBuffers Property</td>
<td>470</td>
</tr>
<tr>
<td>InsertAllSetFields Property</td>
<td>471</td>
</tr>
<tr>
<td>LocalMasterDetail Property</td>
<td>471</td>
</tr>
<tr>
<td>LongStrings Property</td>
<td>472</td>
</tr>
<tr>
<td>MasterFields Nullable Property</td>
<td>472</td>
</tr>
<tr>
<td>NumberRange Property</td>
<td>473</td>
</tr>
<tr>
<td>QueryRecCount Property</td>
<td>473</td>
</tr>
<tr>
<td>QuoteNames Property</td>
<td>474</td>
</tr>
<tr>
<td>RemoveOnRefresh Property</td>
<td>474</td>
</tr>
<tr>
<td>RequiredFields Property</td>
<td>475</td>
</tr>
<tr>
<td>ReturnParams Property</td>
<td>475</td>
</tr>
<tr>
<td>XIContents</td>
<td>476</td>
</tr>
</tbody>
</table>
SetFieldsReadonly Property ................................................................. 476
StrictUpdate Property ........................................................................ 477
TrimFixedChar Property .................................................................... 477
UpdateAllFields Property .................................................................... 478
UpdateBatchSize Property .................................................................... 478
TDAEncryption Class ........................................................................... 479
Members .......................................................................................... 479
Properties ......................................................................................... 479
Encryptor Property .............................................................................. 480
Fields Property ................................................................................ 481
TDAMapRule Class ............................................................................ 481
Members .......................................................................................... 482
Properties ......................................................................................... 482
DBLengthMax Property ....................................................................... 484
DBLengthMin Property ....................................................................... 484
DBScaleMax Property ....................................................................... 485
DBScaleMin Property ....................................................................... 485
DBType Property ............................................................................... 486
FieldLength Property ....................................................................... 486
FieldName Property .......................................................................... 487
FieldScale Property ........................................................................ 487
FieldType Property ........................................................................... 487
IgnoreErrors Property ....................................................................... 487
TDAMapRules Class .......................................................................... 488
Members .......................................................................................... 488
Properties ......................................................................................... 489
IgnoreInvalidRules Property ................................................................. 489
TDAMetaData Class ............................................................................ 490
Members .......................................................................................... 491
Properties ......................................................................................... 494
Connection Property ......................................................................... 495
MetaDataKind Property ..................................................................... 496
Restrictions Property ......................................................................... 497
Methods .......................................................................................... 498
GetMetaDataKinds Method .................................................................. 500
GetRestrictions Method ................................................................... 501
TDAParam Class ............................................................................... 501
Members .......................................................................................... 502
Properties ......................................................................................... 504
AsBlob Property ............................................................................... 505
AsBlobRef Property ............................................................................ 506
AsFloat Property ............................................................................... 506
AsInteger Property ............................................................................ 507
AsLargeInt Property ......................................................................... 507
AsMemo Property ............................................................................... 508
AsMemoRef Property ........................................................................... 508
AsSQLTime_stamp Property ................................................................. 509
AsString Property .............................................................................. 509
AsWideString Property ..................................................................... 510
DataType Property ............................................................................ 510
IsNull Property ................................................................................ 511
ParamType Property .......................................................................... 511
Size Property .................................................................................. 511
Value Property ................................................................................ 512

© 2021 Devart
<table>
<thead>
<tr>
<th>Class</th>
<th>Members</th>
<th>Properties</th>
<th>Methods</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAParams Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDATransaction Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMacro Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMACSs Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Properties**

- `Active Property`  
- `AsDateTime Property`  
- `AsFloat Property`  
- `AsInteger Property`  
- `AsString Property`  
- `Name Property`  
- `Value Property`  

**Methods**

- `AssignField Method`  
- `AssignFieldValue Method`  
- `LoadFromFile Method`  
- `LoadFromStream Method`  
- `SetBlobData Method`  
- `SetBlobData Method`  
- `SetBlobData Method`  
- `MacroByName Method`  
- `Expand Method`  
- `AssignValues Method`  
- `Items Property(Indexer)`  
- `AsInteger Property`  
- `AsFloat Property`  
- `AsDateTime Property`  
- `Active Property`  
- `OnRollbackRetaining Event`  
- `OnError Event`  
- `OnCommit Event`  
- `StartTransaction Method`  
- `Commit Method`  
- `Rollback Method`  
- `FindParam Method`  
- `ParamByName Method`  
- `Items Property(Indexer)`  
- `SetBlobData Method`  
- `LoadFromStream Method`  
- `LoadFromFile Method`  
- `AssignFieldValue Method`  
- `AssignField Method`  
- `Scan Method`  
- `Equals Method`  
- `MacroByName Method`  
- `Scan Method`  
- `SetBlobData Method`  
- `SetBlobData Method`  
- `SetBlobData Method`  
- `SetBlobData Method`  

© 2021 Devart
ConnectionLifetime Property ................................................................. 544
MaxPoolSize Property ............................................................................ 544
MinPoolSize Property ............................................................................. 545
Validate Property .................................................................................. 545
TSmartFetchOptions Class ...................................................................... 546
Members ................................................................................................. 546
Properties ............................................................................................... 547
Enabled Property .................................................................................... 547
LiveBlock Property .................................................................................. 548
PrefetchedFields Property ....................................................................... 548
SQLGetKeyValues Property ..................................................................... 549
Types ........................................................................................................ 549
TAfterExecuteEvent Procedure Reference ............................................. 550
TAfterFetchEvent Procedure Reference .................................................. 551
TBeforeFetchEvent Procedure Reference .............................................. 551
TConnectionLostEvent Procedure Reference ....................................... 552
TDAConnectionErrorEvent Procedure Reference .............................. 552
TDATransactionErrorEvent Procedure Reference ............................. 553
TRefreshOptions Set ................................................................................ 553
TUpdateExecuteEvent Procedure Reference ........................................ 554
Enumerations ............................................................................................ 554
 TLabelSet Enumeration ............................................................................... 555
 TLockMode Enumeration ........................................................................... 555
 TRefreshOption Enumeration .................................................................. 556
 TRetryMode Enumeration ......................................................................... 556
Variables .................................................................................................... 557
 BaseSQLOldBehavior Variable .............................................................. 558
 ChangeCursor Variable ............................................................................ 558
 SQLGeneratorCompatibility Variable .................................................... 558
12 MemData ............................................................................................... 559
Classes ....................................................................................................... 561
TAttribute Class ..................................................................................... 561
Members ................................................................................................. 562
Properties ............................................................................................... 563
AttributeNo Property ............................................................................... 564
DataType Property .................................................................................... 564
Length Property ........................................................................................ 565
ObjectType Property .................................................................................. 566
Offset Property ......................................................................................... 566
Owner Property ........................................................................................ 566
Scale Property .......................................................................................... 567
Size Property ............................................................................................ 567
TDBlob Class ........................................................................................... 568
Members ................................................................................................. 569
Properties ............................................................................................... 570
AsString Property .................................................................................... 571
AsStringWideProperty ........................................................................... 571
IsUnicode Property .................................................................................. 572
Size Property ............................................................................................ 572
Methods .................................................................................................. 573
Assign Method ......................................................................................... 574
Clear Method ........................................................................................... 574
LoadFromFile Method .............................................................................. 575

© 2021 Devart
### MemDS

<table>
<thead>
<tr>
<th>Classes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMemDataSet Class</td>
<td>601</td>
</tr>
<tr>
<td>Members</td>
<td>602</td>
</tr>
<tr>
<td>Properties</td>
<td>605</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCompressedBlob Class</td>
<td>579</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Members</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td></td>
</tr>
<tr>
<td>Compressed Property</td>
<td>582</td>
</tr>
<tr>
<td>CompressedSize Property</td>
<td>583</td>
</tr>
</tbody>
</table>

| TDBObject Class | 583 |

| Members | 584 |
| Properties | |
| AttributeCount Property | 587 |
| Attributes Property(Indexer) | 588 |
| DataType Property | 588 |
| Size Property | 589 |

| Methods | 589 |
| FindAttribute Method | 590 |

| TSharedObject Class | 591 |

| Members | 591 |
| Properties | |
| RefCount Property | 592 |

| Methods | 593 |
| AddRef Method | 593 |
| Release Method | 594 |

| Types | 594 |
| TLlocateExOptions Set | 595 |
| TUpdateRecKinds Set | 595 |

| Enumerations | 595 |
| TCompressBlobMode Enumeration | 596 |
| TConnLostCause Enumeration | 597 |
| TDANumericType Enumeration | 598 |
| TLlocateExOption Enumeration | 598 |
| TSortType Enumeration | 599 |
| TUpdateRecKind Enumeration | 600 |

13 MemDS ......................................................................................................................... 600
14 PqAccess

Classes

TCustomPgDataSet Class

Methods

UpdatesPending Property........................................................................................................... 611
ApplyRange Method................................................................................................................... 612
ApplyUpdates Method.................................................................................................................. 614
ApplyUpdates Method.................................................................................................................. 615
ApplyUpdates Method.................................................................................................................. 615
ApplyUpdates Method.................................................................................................................. 617
CancelRange Method.................................................................................................................. 618
CancelUpdates Method............................................................................................................... 618
CommitUpdates Method.............................................................................................................. 619
DeferredPost Method................................................................................................................ 620
EditRangeEnd Method.............................................................................................................. 620
EditRangeStart Method............................................................................................................ 621
GetBlob Method....................................................................................................................... 622
GetBlob Method....................................................................................................................... 622
GetBlob Method....................................................................................................................... 623
Locate Method........................................................................................................................ 623
Locate Method........................................................................................................................ 624
Locate Ex Method..................................................................................................................... 624
Locate Ex Method..................................................................................................................... 626
Locate Ex Method..................................................................................................................... 626
Locate Ex Method..................................................................................................................... 627
Prepare Method......................................................................................................................... 628
RestoreUpdates Method............................................................................................................ 628
RevertRecord Method.............................................................................................................. 628
SaveToXML Method.................................................................................................................. 629
SaveToXML Method.................................................................................................................. 630
SaveToXML Method.................................................................................................................. 631
SetRange Method...................................................................................................................... 632
SetRangeEnd Method............................................................................................................... 633
SetRangeStart Method............................................................................................................ 634
UnPrepare Method.................................................................................................................... 634
UpdateResult Method.............................................................................................................. 635
UpdateStatus Method.............................................................................................................. 636
Events

OnUpdateError Event............................................................................................................. 637
OnUpdateRecord Event............................................................................................................. 637

Variables

DoNotRaiseExeptionOnUsFail Variable.................................................................................... 639
SendDataSetChangeEventAfterOpen Variable.......................................................................... 640

© 2021 Devart
TCustomPgTimeStampField Class

<table>
<thead>
<tr>
<th>Methods</th>
<th>669</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateProcCall Method</td>
<td></td>
</tr>
<tr>
<td>FindParam Method</td>
<td>670</td>
</tr>
<tr>
<td>GetPgC cursor Method</td>
<td>671</td>
</tr>
<tr>
<td>GetPgDate Method</td>
<td>671</td>
</tr>
<tr>
<td>GetPgInterval Method</td>
<td>671</td>
</tr>
<tr>
<td>GetPgLargeObject Method</td>
<td>672</td>
</tr>
<tr>
<td>GetPgRow Method</td>
<td>673</td>
</tr>
<tr>
<td>GetPgTime Method</td>
<td>673</td>
</tr>
<tr>
<td>GetPgTimeMethod</td>
<td>674</td>
</tr>
<tr>
<td>GetPgTimeStamp Method</td>
<td>674</td>
</tr>
<tr>
<td>OpenNext Method</td>
<td>675</td>
</tr>
<tr>
<td>ParamByName Method</td>
<td>675</td>
</tr>
</tbody>
</table>

TCustomPgTable Class

<table>
<thead>
<tr>
<th>Members</th>
<th>676</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>677</td>
</tr>
<tr>
<td>Overload Property</td>
<td>686</td>
</tr>
<tr>
<td>Methods</td>
<td>690</td>
</tr>
<tr>
<td>ExecProc Method</td>
<td>691</td>
</tr>
<tr>
<td>PrepareSQL Method</td>
<td>695</td>
</tr>
</tbody>
</table>

TCustomPgStoredProc Class

<table>
<thead>
<tr>
<th>Methods</th>
<th>696</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateTransaction Method</td>
<td>695</td>
</tr>
<tr>
<td>BreakExec Method</td>
<td>696</td>
</tr>
</tbody>
</table>

TPgConnection Class

<table>
<thead>
<tr>
<th>Properties</th>
<th>713</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionTimeout Property</td>
<td>714</td>
</tr>
<tr>
<td>Database Property</td>
<td></td>
</tr>
<tr>
<td>Options Property</td>
<td></td>
</tr>
<tr>
<td>Password Property</td>
<td></td>
</tr>
<tr>
<td>Port Property</td>
<td></td>
</tr>
<tr>
<td>ProcessID Property</td>
<td></td>
</tr>
<tr>
<td>ProtocolVersion Property</td>
<td></td>
</tr>
<tr>
<td>Schema Property</td>
<td></td>
</tr>
<tr>
<td>Server Property</td>
<td></td>
</tr>
<tr>
<td>ServerVersion Property</td>
<td></td>
</tr>
<tr>
<td>ServerVersionFull Property</td>
<td></td>
</tr>
<tr>
<td>SSLOptions Property</td>
<td></td>
</tr>
<tr>
<td>Username Property</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>723</th>
</tr>
</thead>
<tbody>
<tr>
<td>BreakExec Method</td>
<td>723</td>
</tr>
<tr>
<td>CreateDataSet Method</td>
<td>730</td>
</tr>
<tr>
<td>CreateMetaData Method</td>
<td>730</td>
</tr>
<tr>
<td>CreateSQL Method</td>
<td>731</td>
</tr>
<tr>
<td>CreateTransaction Method</td>
<td>731</td>
</tr>
<tr>
<td>GetRow Type Method</td>
<td>732</td>
</tr>
<tr>
<td>GetRow Type Method</td>
<td>732</td>
</tr>
<tr>
<td>GetRowType Method</td>
<td>732</td>
</tr>
<tr>
<td>GetRowType Method</td>
<td>732</td>
</tr>
<tr>
<td>ReleaseSavepoint Method</td>
<td>734</td>
</tr>
<tr>
<td>RollbackToSavepoint Method</td>
<td>735</td>
</tr>
</tbody>
</table>
PostgreSQL Data Access Components

- TPgDateField Class
- TPgDataSource Class
- TPgDataSetOptions Class
- TPgConnectionSSLOptions Class
- TPgConnectionOptions Class

Properties

Members

Events

AsPgDate Property
UnknownAsString Property
PrepareUpdateSQL Property
OIDAsInt Property
FullRefresh Property
EnableBCD Property
DistinctParams Property
CacheBlobs Property
Mode Property
AsCursor Property

TPgConnectionSSLOptions Class

Members

Properties

ApplicationName Property
Charset Property
EnableBCD Property
EnableComposites Property
EnableDomains Property
EnableFMTBCD Property
EnableGeometrics Property
EnablePgTimeStamps Property
ImmediateNotices Property
IPVersion Property
MessagesCharset Property
UseUnicode Property

TPgConnectionOptions Class

Members

Properties

AutoDeleteBlob Property
CacheBlobs Property
CursorWithHold Property
DeferredBlobRead Property
DistinctParams Property
EnableBCD Property
EnableFMTBCD Property
ExtendedFieldsInfo Property
FullRefresh Property
OIDAsInt Property
PrepareUpdateSQL Property
SetEmptyStrToNull Property
UnknownAsString Property
UnpreparedExecute Property
UseParamTypes Property

TPgDataSource Class

Members

TPgDateField Class

Members

Properties

AsPgDate Property
TPgEncryptor Class .......................................................................................................................... 773
Members ........................................................................................................................................ 773
TPgGeometricField Class .................................................................................................................. 774
Members ........................................................................................................................................ 774
Properties ....................................................................................................................................... 775
AsPgGeometric Property .................................................................................................................. 775
TPgIntervalField Class ...................................................................................................................... 776
Members ........................................................................................................................................ 776
Properties ....................................................................................................................................... 776
AsPgInterval Property ....................................................................................................................... 777
TPgLargeObjectClass .......................................................................................................................... 777
Members ......................................................................................................................................... 778
Properties ....................................................................................................................................... 780
Connection Property ......................................................................................................................... 781
TPgLargeObjectField Class .................................................................................................................. 781
Members ......................................................................................................................................... 782
Properties ....................................................................................................................................... 782
AsLargeObject Property .................................................................................................................... 783
TPgMetaData Class .............................................................................................................................. 783
Members ......................................................................................................................................... 784
TPgParam Class .................................................................................................................................. 787
Members ......................................................................................................................................... 788
Properties ....................................................................................................................................... 790
AsPgDate Property ............................................................................................................................... 792
AsPgInterval Property .......................................................................................................................... 792
AsPgTime Property ............................................................................................................................... 793
AsPgTimeStamp Property ..................................................................................................................... 793
TPgParams Class .................................................................................................................................. 793
Members ......................................................................................................................................... 794
Properties ....................................................................................................................................... 795
Items Property (Indexer) ...................................................................................................................... 795
TPgQuery Class ................................................................................................................................... 796
Members ......................................................................................................................................... 797
Properties ....................................................................................................................................... 806
FetchAll Property ................................................................................................................................. 811
LockMode Property ................................................................................................................................. 811
UpdatingTable Property ....................................................................................................................... 812
TPgSQLClass ....................................................................................................................................... 813
Members ......................................................................................................................................... 814
Properties ....................................................................................................................................... 816
CommandTimeout Property .................................................................................................................. 818
Connection Property ............................................................................................................................. 818
LastInsertOID Property ......................................................................................................................... 819
Params Property .................................................................................................................................. 819
UnpreparedExecute Property ................................................................................................................. 820
UseParamTypes Property ..................................................................................................................... 820
Methods ........................................................................................................................................... 821
FindParam Method ............................................................................................................................... 822
ParamByName Method ......................................................................................................................... 823
TPgStoredProcClass ............................................................................................................................ 823
Members ......................................................................................................................................... 824
Properties ....................................................................................................................................... 834
CommandTimeout Property .................................................................................................................. 838
LockMode Property ............................................................................................................................... 838
### PostgreSQL Data Access Components

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgConnectionPoolManager Class</td>
<td>860</td>
</tr>
<tr>
<td>TPgCursor Class</td>
<td>864</td>
</tr>
<tr>
<td>TPgAlerter Class</td>
<td>868</td>
</tr>
<tr>
<td>TSslMode Enumeration</td>
<td>866</td>
</tr>
<tr>
<td>TPgNoticeEvent Procedure Reference</td>
<td>866</td>
</tr>
<tr>
<td>TPgNotificationEvent Procedure Reference</td>
<td>866</td>
</tr>
<tr>
<td>TPgTransaction Class</td>
<td>864</td>
</tr>
<tr>
<td>TPgTimeField Class</td>
<td>861</td>
</tr>
<tr>
<td>TPgTimeField Class</td>
<td>862</td>
</tr>
<tr>
<td>TPgTimeField Class</td>
<td>863</td>
</tr>
<tr>
<td>TPgUpdateSQL Class</td>
<td>864</td>
</tr>
<tr>
<td>TPgUpdateSQL Class</td>
<td>864</td>
</tr>
<tr>
<td>TPgTimeStampField Class</td>
<td>864</td>
</tr>
<tr>
<td>TPgTimeStampField Class</td>
<td>865</td>
</tr>
<tr>
<td>TPgEnumEnumeration</td>
<td>867</td>
</tr>
<tr>
<td>TPgEnumEnumeration</td>
<td>867</td>
</tr>
<tr>
<td>TPgEnumEnumeration</td>
<td>868</td>
</tr>
<tr>
<td>TPgEnumEnumeration</td>
<td>868</td>
</tr>
<tr>
<td>TPgEnumEnumeration</td>
<td>869</td>
</tr>
<tr>
<td>TPgEnumEnumeration</td>
<td>869</td>
</tr>
</tbody>
</table>

#### Members

<table>
<thead>
<tr>
<th>Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>StoredProcName Property</td>
<td>839</td>
</tr>
<tr>
<td>TPgTable Class</td>
<td>839</td>
</tr>
<tr>
<td>Members</td>
<td>840</td>
</tr>
<tr>
<td>Properties</td>
<td>850</td>
</tr>
<tr>
<td>FetchAll Property</td>
<td>854</td>
</tr>
<tr>
<td>LockMode Property</td>
<td>855</td>
</tr>
<tr>
<td>OrderFields Property</td>
<td>856</td>
</tr>
<tr>
<td>TableName Property</td>
<td>856</td>
</tr>
<tr>
<td>TPgCursor Class</td>
<td>857</td>
</tr>
<tr>
<td>Members</td>
<td>857</td>
</tr>
<tr>
<td>Properties</td>
<td>858</td>
</tr>
<tr>
<td>AsPgTime Property</td>
<td>858</td>
</tr>
<tr>
<td>TPgTimeStampField Class</td>
<td>859</td>
</tr>
<tr>
<td>Members</td>
<td>859</td>
</tr>
<tr>
<td>Properties</td>
<td>859</td>
</tr>
<tr>
<td>AsPgTimeStamp Property</td>
<td>860</td>
</tr>
<tr>
<td>TPgTransaction Class</td>
<td>861</td>
</tr>
<tr>
<td>Members</td>
<td>862</td>
</tr>
<tr>
<td>Properties</td>
<td>863</td>
</tr>
<tr>
<td>IsolationLevel Property</td>
<td>863</td>
</tr>
<tr>
<td>TPgUpdateSQL Class</td>
<td>864</td>
</tr>
<tr>
<td>Members</td>
<td>864</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnEvent Event</td>
<td>870</td>
</tr>
<tr>
<td>SendEvent Method</td>
<td>871</td>
</tr>
<tr>
<td>Events</td>
<td>872</td>
</tr>
<tr>
<td>Methods</td>
<td>872</td>
</tr>
<tr>
<td>SendEvent Method</td>
<td>873</td>
</tr>
<tr>
<td>Events</td>
<td>873</td>
</tr>
<tr>
<td>OnEvent Event</td>
<td>874</td>
</tr>
</tbody>
</table>

#### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>874</td>
</tr>
<tr>
<td>OnEvent Event</td>
<td>875</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 PgAlerter</td>
<td>869</td>
</tr>
<tr>
<td>TPgAlerter Class</td>
<td>869</td>
</tr>
<tr>
<td>Members</td>
<td>870</td>
</tr>
<tr>
<td>Properties</td>
<td>871</td>
</tr>
<tr>
<td>Events Property</td>
<td>872</td>
</tr>
<tr>
<td>Methods</td>
<td>872</td>
</tr>
<tr>
<td>SendEvent Method</td>
<td>873</td>
</tr>
<tr>
<td>SendEvent Method</td>
<td>873</td>
</tr>
<tr>
<td>Events</td>
<td>874</td>
</tr>
<tr>
<td>OnEvent Event</td>
<td>875</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 PgClasses</td>
<td>875</td>
</tr>
<tr>
<td>TPgCursor Class</td>
<td>876</td>
</tr>
<tr>
<td>Members</td>
<td>877</td>
</tr>
<tr>
<td>Properties</td>
<td>877</td>
</tr>
<tr>
<td>State Property</td>
<td>878</td>
</tr>
<tr>
<td>Enumerations</td>
<td>878</td>
</tr>
<tr>
<td>TPgProtocolVersion Enumeration</td>
<td>879</td>
</tr>
<tr>
<td>TSslMode Enumeration</td>
<td>879</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17 PgConnectionPool</td>
<td>880</td>
</tr>
<tr>
<td>TPgConnectionPoolManager Class</td>
<td>880</td>
</tr>
</tbody>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Classes</th>
<th>Members</th>
<th>18 PgDacVcl</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgConnectDialog Class</td>
<td>Properties</td>
<td>881</td>
</tr>
<tr>
<td>Members</td>
<td>Connection Property</td>
<td>882</td>
</tr>
<tr>
<td>Properties</td>
<td>DatabaseLabel Property</td>
<td>886</td>
</tr>
<tr>
<td></td>
<td>PortLabel Property</td>
<td>886</td>
</tr>
<tr>
<td></td>
<td>Show Database Property</td>
<td>887</td>
</tr>
<tr>
<td></td>
<td>Show Port Property</td>
<td>888</td>
</tr>
<tr>
<td>Classes</td>
<td>Members</td>
<td>19 PgDump</td>
</tr>
<tr>
<td>TPgDump Class</td>
<td>Properties</td>
<td>888</td>
</tr>
<tr>
<td>Members</td>
<td>Mode Property</td>
<td>890</td>
</tr>
<tr>
<td>Properties</td>
<td>ObjectTypes Property</td>
<td>892</td>
</tr>
<tr>
<td></td>
<td>Options Property</td>
<td>893</td>
</tr>
<tr>
<td></td>
<td>SchemaNames Property</td>
<td>894</td>
</tr>
<tr>
<td>TPgDumpOptions Class</td>
<td>Members</td>
<td>895</td>
</tr>
<tr>
<td>Members</td>
<td>CreateConstraints Property</td>
<td>896</td>
</tr>
<tr>
<td>Properties</td>
<td>Position Property</td>
<td>897</td>
</tr>
<tr>
<td></td>
<td>LineNumber Property</td>
<td>898</td>
</tr>
<tr>
<td>Types</td>
<td>TPgDumpMode Enumeration</td>
<td>899</td>
</tr>
<tr>
<td>TPgDumpObjects Set.</td>
<td>TPgDumpObject Enumeration</td>
<td></td>
</tr>
<tr>
<td>Enumerations</td>
<td>Members</td>
<td></td>
</tr>
<tr>
<td>TPgSeverity Enumeration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 PgError</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td>Members</td>
<td></td>
</tr>
<tr>
<td>EPgError Class</td>
<td>Properties</td>
<td>900</td>
</tr>
<tr>
<td>Members</td>
<td>CallStack Property</td>
<td>901</td>
</tr>
<tr>
<td>Properties</td>
<td>DetailMsg Property</td>
<td>902</td>
</tr>
<tr>
<td></td>
<td>ErrorCode Property</td>
<td>903</td>
</tr>
<tr>
<td></td>
<td>FileName Property</td>
<td>904</td>
</tr>
<tr>
<td></td>
<td>Hint Property</td>
<td>905</td>
</tr>
<tr>
<td></td>
<td>LineNumber Property</td>
<td>906</td>
</tr>
<tr>
<td></td>
<td>Connection Property</td>
<td>907</td>
</tr>
<tr>
<td>Enforcements</td>
<td>Members</td>
<td></td>
</tr>
<tr>
<td>TPgSeverity Enumeration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21 PgLoader</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td>Members</td>
<td></td>
</tr>
<tr>
<td>TPgLoader Class</td>
<td>Properties</td>
<td>908</td>
</tr>
<tr>
<td>Members</td>
<td>BufferSize Property</td>
<td>909</td>
</tr>
<tr>
<td>Properties</td>
<td>Columns Property</td>
<td>911</td>
</tr>
<tr>
<td></td>
<td>Connection Property</td>
<td>912</td>
</tr>
</tbody>
</table>
PostgreSQL Data Access Components

Options Property ................................................................. 913
TableName Property ............................................................... 913
TextMode Property ............................................................... 914

Events .............................................................................. 914
OnGetColumnData Event ....................................................... 915
OnPutData Event ................................................................. 915

TPgLoaderColumn Class ....................................................... 915
Members ........................................................................ 916
Properties ....................................................................... 917

Row TypeName Property ....................................................... 917

TPgLoaderOptions Class ...................................................... 918
Members ........................................................................ 918
Properties ....................................................................... 919

BufferSize Property ........................................................... 919
QuoteNames Property ........................................................ 920
TextMode Property ........................................................... 920
UseBlankValues Property ................................................... 921

TextMode Property ........................................................... 922

22 PgObjects ...................................................................... 921

Classes ............................................................................ 923

TCustomPgTimeStamp Class .................................................. 924
Members ........................................................................ 925

AsDateTime Property .......................................................... 926
AsSQLTimeStamp Property .................................................. 926
AsString Property ............................................................. 928
Days Property ................................................................... 929
HasTimeZone Property ......................................................... 929
IsInfinity Property .............................................................. 930
IsNegInfinity Property ........................................................ 930
IsPosInfinity Property ........................................................ 931

Ticks Property ................................................................... 931

TimeZoneOffset Property ..................................................... 932

Methods ........................................................................... 932

Assign Method ................................................................. 933
Compare Method ............................................................... 934
DecodeDate Method ........................................................... 934
DecodeDateTime Method ...................................................... 935
DecodeTime Method ........................................................... 936

EncodeDate Method ........................................................... 937
EncodeDateTime Method ....................................................... 937
EncodeTime Method ........................................................... 938

TPgAttribute Class ............................................................ 939
Members ........................................................................ 939

TPgBox Class .................................................................. 940
Members ........................................................................ 941

Properties ....................................................................... 942

LowerLeft Property ........................................................... 942
UpperRight Property ........................................................ 943

TPgCircle Class ................................................................. 944
Members ........................................................................ 944

Properties ....................................................................... 944

Center Property ............................................................... 945

Radius Property ............................................................... 945

TPgDate Class ................................................................. 946
<table>
<thead>
<tr>
<th>Class</th>
<th>Members</th>
<th>Properties</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgGeometric Class</td>
<td>946</td>
<td>948</td>
<td>949</td>
</tr>
<tr>
<td>TPgPolygon Class</td>
<td>950</td>
<td>951</td>
<td>952</td>
</tr>
<tr>
<td>TPgPointsArray Class</td>
<td>954</td>
<td>955</td>
<td>956</td>
</tr>
<tr>
<td>TPgPoint Class</td>
<td>958</td>
<td>960</td>
<td>961</td>
</tr>
<tr>
<td>TPgPath Class</td>
<td>964</td>
<td>965</td>
<td>966</td>
</tr>
<tr>
<td>TPgLSeg Class</td>
<td>969</td>
<td>970</td>
<td>971</td>
</tr>
<tr>
<td>TPgInterval Class</td>
<td>973</td>
<td>974</td>
<td>975</td>
</tr>
<tr>
<td>TPgRow Class</td>
<td>978</td>
<td>980</td>
<td>981</td>
</tr>
</tbody>
</table>
23 PgScript ................................................................. 1007
   Classes ........................................................................ 1008
      TPgScript Class ........................................................ 1008
         Members ............................................................... 1008
         Properties ............................................................ 1009
            Connection Property ........................................ 1010
            TZ Offset Property ........................................... 1012

24 PostgreSQLMonitor .................................................. 1012
   Classes ........................................................................ 1012
      PostgreSQLMonitor Class ........................................ 1013
         Members ............................................................... 1014

25 VirtualDataSet ........................................................ 1014
   Classes ........................................................................ 1015
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomVirtualDataSet Class</td>
<td>1016</td>
</tr>
<tr>
<td>Members</td>
<td>1016</td>
</tr>
<tr>
<td>TVirtualDataSet Class</td>
<td>1019</td>
</tr>
<tr>
<td>Members</td>
<td>1020</td>
</tr>
<tr>
<td>Types</td>
<td>1023</td>
</tr>
<tr>
<td>TOnDeleteRecordEvent Procedure Reference</td>
<td>1023</td>
</tr>
<tr>
<td>TOnGetFieldValueEvent Procedure Reference</td>
<td>1024</td>
</tr>
<tr>
<td>TOnGetRecordCountEvent Procedure Reference</td>
<td>1024</td>
</tr>
<tr>
<td>TOnModifyRecordEvent Procedure Reference</td>
<td>1025</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td>1025</td>
</tr>
<tr>
<td>TVirtualTable Class</td>
<td>1026</td>
</tr>
<tr>
<td>Members</td>
<td>1026</td>
</tr>
<tr>
<td>Properties</td>
<td>1030</td>
</tr>
<tr>
<td>DefaultSortType Property</td>
<td>1031</td>
</tr>
<tr>
<td>Methods</td>
<td>1031</td>
</tr>
<tr>
<td>Assign Method</td>
<td>1034</td>
</tr>
<tr>
<td>LoadFromFile Method</td>
<td>1034</td>
</tr>
<tr>
<td>LoadFromStream Method</td>
<td>1035</td>
</tr>
</tbody>
</table>
1 What's New

02-Mar-21 New Features in PgDAC 6.4:
- PostgreSQL 13 is supported
- RAD Studio 10.4.2 Sydney is supported
- macOS 11 Big Sur is supported
- iOS 14 is supported
- Android 11 is supported
- Work in a multi-threaded environment through a single connection is supported
- The MultipleConnections option in the Connection component is added
- Performance of batch operations is improved
- Performance of the FindFirst, FindNext, FindLast, and FindPrior methods is improved

26-Aug-20 New Features in PgDAC 6.3:
- Lazarus 2.0.10 and FPC 3.2.0 are supported
- Performance of Batch Insert, Update, and Delete operations is improved

01-Jun-20 New Features in PgDAC 6.2:
- RAD Studio 10.4 Sydney is supported
- Lazarus 2.0.8 is supported
- macOS 64-bit in Lazarus is supported
- The Line geometric type is supported

26-Nov-2019 New Features in PgDAC 6.1:
- Android 64-bit is supported
- PostgreSQL 12 is supported
- OpenSSL 1.1 library is supported
- Now Trial edition for macOS and Linux is fully functional

22-Jul-2019 New Features in PgDAC 6.0:
- macOS 64-bit is supported
- Release 2 for RAD Studio 10.3 Rio, Delphi 10.3 Rio, and C++Builder 10.3 Rio is now required
- TPgConnectionSSLOptions.IgnoreServerCertificateInsecurity property is added
24-Jun-2019 New Features in PgDAC 5.4:
- Lazarus 2.0.2 is supported
- The pmAuto value for the ProtocolVersion property is added
- Now ProtocolVersion is set to pmAuto by default, which significantly improved performance
- Possibility to use function calls in batch operations is added
- The mVerifyCA and smVerifyFull options for the SSLOptions.Mode property of the TPgConnection component are added
- The DefaultSortType property for TVirtualTable is added
- Performance of the SaveToFile/LoadFromFile methods of TVirtualTable is significantly increased

26-Nov-2018 New Features in PgDAC 5.3:
- RAD Studio 10.3 Rio is supported
- PostgreSQL 11 is supported
- Support of UPPER and LOWER functions for Unified SQL is added

09-Jul-2018 New Features in PgDAC 5.2:
- Lazarus 1.8.4 is supported
- Performance of batch operations is improved
- Support for HTTP/HTTPS tunnel is added
- Demo projects for IntraWeb 14 are added

19-Sep-2017 New Features in PgDAC 5.1:
- SSPI authentication is supported
- Processing GUID data type for the TGuidField class is improved

05-Apr-2017 New Features in PgDAC 5.0:
- RAD Studio 10.2 Tokyo is supported
- Linux in RAD Studio 10.2 Tokyo is supported
- Lazarus 1.6.4 and Free Pascal 3.0.2 is supported

25-Apr-16 New Features in PgDAC 4.7:
- RAD Studio 10.1 Berlin is supported
- Lazarus 1.6 and FPC 3.0.0 is supported
- PostgreSQL 9.5 is supported
• Support for the BETWEEN statement in TDADataSet.Filter is added
• A MessageCharset option in connection options is added
• Data Type Mapping performance is improved
• RepeatableRead and ReadUncommitted transaction isolation levels are added
• Performance of TDALoader on loading data from TDataSet is improved

09-Sep-15 New Features in PgDAC 4.6:
• RAD Studio 10 Seattle is supported
• INSERT, UPDATE and DELETE batch operations are supported
• Support of bit and bit varying data types is improved
• Now Trial for Win64 is a fully functional Professional Edition

14-Apr-15 New Features in PgDAC 4.5:
• RAD Studio XE8 is supported
• AppMethod is supported
• PostgreSQL 9.4 is supported

15-Sep-14 New Features in PgDAC 4.4:
• RAD Studio XE7 is supported
• Lazarus 1.2.4 is supported
• Demo projects for FastReport 5 are added
• The TCustomDADataSet.GetKeyFieldNames method is added
• The ConstraintColumns metadata kind for the TDAMetadata component is added

29-Apr-14 New Features in PgDAC 4.3:
• Delphi XE6 is supported
• Android in C++Builder XE6 is supported
• Lazarus 1.2.2 and FPC 2.6.4 is supported
• SmartFetch mode for TDataSet descendants is added
• Now update queries inside TDataSet descendants have correct owner
• The TPgDataSetOptions.MasterDetailNullable property is added

25-Dec-13 New Features in PgDAC 4.2:
• iOS in C++Builder XE5 is supported
• RAD Studio XE5 Update 2 is now required
• Now .obj and .o files are supplied for C++Builder
• A list of available Charsets in TPgConnection at design-time is added
• Default charset detecting for Windows is added
• Compatibility of migrating floating-point fields from other components is improved

18-Sep-13 New Features in PgDAC 4.1:
• RAD Studio XE5 is supported
• Application development for Android is supported
• Lazarus 1.0.12 is supported
• Performance is improved
• Automatic checking for new versions is added
• Flexible management of conditions in the WHERE clause is added
• The possibility to use conditions is added
• PostgreSQL 9.3 is supported
• IPv6 protocol support is added
• Support of the IN keyword in the TDataSet.Filter property is added
• Like operator behaviour when used in the Filter property is now similar to TClientDataSet
• The possibility to use ranges is added
• The Ping method for the Connection component is added
• The AllowImplicitConnect option for the Connection component is added
• The SQLRecCount property for the Query and StoredProc components is added
• The ScanParams property for the Script component is added
• The RowsAffected property for the Script component is added
• The EnableDomains option is added for TPgConnection
• ConnectionTimeout is now used when disconnecting after connection loss
• The TPgTable.TableName and TPgStoredProc.StoredProcName property editors are improved

25-Apr-13 New Features in PgDAC 4.0:
• Rad Studio XE4 is supported
• NEXTGEN compiler is supported
• Application development for iOS is supported
• FPC 2.6.2 and Lazarus 1.0.8 are supported
• Connection string support is added
• Possibility to encrypt entire tables and datasets is added
• Possibility to determine if data in a field is encrypted is added
• Support of TimeStamp, Single and Extended fields in VirtualTable is added
• Support for custom mapping of numeric fields with BCD and FmtBCD types is added

12-Dec-12 New Features in PgDAC 3.6:
• Rad Studio XE3 Update 1 is now required
• C++Builder 64-bit for Windows is supported

05-Sep-12 New Features in PgDAC 3.5:
• Rad Studio XE3 is supported
• Windows 8 is supported

23-Nov-11 New Features in PgDAC 3.1:
• Update 2 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
• Mac OS X and iOS in RAD Studio XE2 is supported
• FireMonkey support is improved
• Lazarus 0.9.30.2 and FPC 2.4.4 are supported
• Mac OS X in Lazarus is supported
• Linux x64 in Lazarus is supported
• FreeBSD in Lazarus is supported
• PostgreSQL 9.1 is supported

15-Sep-11 New Features in PostgreSQL Data Access Components 3.00:
• Embarcadero RAD Studio XE2 is supported
• Application development for 64-bit Windows is supported
• FireMonkey application development platform is supported
• Support of master/detail relationship for TVirtualTable is added
• OnProgress event in TVirtualTable is added
• TDADatasetOptions.SetEmptyStrToNull property that allows inserting NULL value instead of empty string is added

28-Apr-11 New Features in PostgreSQL Data Access Components 2.20:
• Lazarus 0.9.30 and FPC 2.4.2 is supported
• Application Name connection option is supported
• Payload parameter for PostgreSQL notification is supported (TPgNotificationEvent changed: EventMessage parameter is added)
26-Jan-11 New Features in PostgreSQL Data Access Components 2.10:
- PostgreSQL 9.0 supported
- Improved performance
- Improved table names detecting inside SQL queries for the UpdatingTable property
- Case sensitive schema name
- Checking that dataset is open on calling the TDataSet.Locate method

13-Sep-10 New Features in PostgreSQL Data Access Components 2.00:
- Embarcadero RAD Studio XE supported
- Support of ONLY lexeme in the FROM statement
- Support for dbMonitor 3
- Added OnStart, OnCommit, OnRollback events to TDataSet
- Ability to lock records in the CachedUpdate mode
- Ability to send call stack information to the dbMonitor component
- Changed the LocateEx method behavior: now LocateEx centers records equal to Locate
- CursorWithHold option for TCustomPgDataSet to use FetchAll=False mode without transaction
- Now Required flag is set for UpdatingTable fields only
- Now the AssignConnect method copies transaction state

10-Sep-09 New Features in PostgreSQL Data Access Components 1.20:
- Embarcadero RAD Studio 2010 supported
- Support for automatic starting a transaction when FetchAll=False
- FullRefresh option for TCustomPgDataSet
- The Disconnected property to TCustomDADataSet
- Distinction between empty string and null value when saving/loading string fields in TVirtualTable
- The UseParamTypes option used to disable automatic detection of parameter types
- Now the value from the master dataset has priority over the DefaultExpression value

02-Apr-09 New Features in PostgreSQL Data Access
Components 1.10:
- Free Pascal under Linux supported
- DMLRefresh supported
- Added NoPreconnect property to TPgScript for executing CONNECT and CREATE DATABASE commands

© 1997-2021 Devart. All Rights Reserved.

2 General Information

This section contains general information about PostgreSQL Server Data Access Components:
- Overview
- Features
- Requirements
- Compatibility
- Using Several DAC Products in One IDE
- Component List
- Hierarchy Chart
- Editions
- Licensing and Subscriptions
- Getting Support

© 1997-2021 Devart. All Rights Reserved.

2.1 Overview

PostgreSQL Data Access Components (PgDAC) is a library of components that provides direct access to PostgreSQL database servers from Delphi, C++Builder and Lazarus (Free Pascal). PgDAC is designed to help programmers develop really lightweight, faster and cleaner PostgreSQL database applications without deploying any additional libraries.

PgDAC is a complete replacement for standard PostgreSQL connectivity solutions and presents an efficient alternative to the Borland Database Engine for access to PostgreSQL.

The PgDAC library is actively developed and supported by the Devart Team. If you have
Advantages of PgDAC Technology

PgDAC works directly through TCP/IP protocol and does not use the PostgreSQL client library. As data is transferred from socket to storage without additional buffers, the PgDAC performance is kept on the highest level. Such technology helps to avoid restrictions and slips of the pqlib library, use features of PostgreSQL backend protocol that are not implemented in it. Also Devart PgDAC offers wide coverage of the PostgreSQL feature set and emphasizes optimized data access strategies.

Wide Coverage of PostgreSQL Features

By providing access to the most advanced database functionality, PgDAC allows developers to harness the full capabilities of the PostgreSQL server and optimize their database applications. PgDAC features complete support of fast record insertion, Asynchronous Notification, PostgreSQL sequences, the possibility to retrieve the last inserted OID value, notices, and more. Get a full list of supported SQL Server features in Features.

Native Connection Options

PgDAC does not require PostgreSQL client software installed what heightens its performance. PgDAC-based database applications are easy to deploy, do not require installation of other data provider layers (such as BDE), and tend to be faster than those that use standard data connectivity solutions.

Optimized Code

The goal of PgDAC is to enable developers to write efficient and flexible database applications. The PgDAC library is implemented using advanced data access algorithms and optimization techniques. Classes and components undergo comprehensive performance tests and are designed to help you write high-performance, lightweight data access layers.

Compatibility with other Connectivity Methods

The PgDAC interface retains compatibility with standard VCL data access components, like BDE.

Development and Support

PgDAC is a PostgreSQL connectivity solution that is actively developed and supported.
PgDAC comes with full documentation, demo projects, and fast (usually within two business days) technical support by the PgDAC development team. Find out more about getting help or submitting feedback and suggestions to PgDAC Development Team in Getting Support.

A description of the PgDAC components is provided in Component List.

### Key Features

The following list describes the main features of PostgreSQL Data Access Components.

- Direct access to server data without using client library. Does not require installation of other data provider layers (such as BDE and ODBC)
- Full support of the latest versions of PostgreSQL Server
- Support for all PostgreSQL Server data types
- Disconnected Model with automatic connection control for working with data offline
- Local Failover for detecting connection loss and implicitly reexecuting certain operations
- All types of local sorting and filtering, including by calculated and lookup fields
- **Automatic data updating** with TPgQuery, TPgTable, and TPgStoredProc components
- Unicode and national charset support
- Supports many PostgreSQL-specific features, such as notifications, notices, and sequences
- Advanced script execution functionality with the TPgScript component
- Support for **using macros** in SQL
- Lets you use Professional Edition of Delphi and C++Builder to develop client/server applications
- Includes annual PgDAC Subscription with Priority Support
- Licensed royalty-free per developer, per team, or per site

The full list of PgDAC features is available in Features.

### How does PgDAC work?

PgDAC connects to PostgreSQL directly without using client software.

In contrast, the Borland Database Engine (BDE) uses several layers to access PostgreSQL and requires additional data access software to be installed on client machines.
2.2 Features

In this topic you will find the complete PgDAC feature list sorted by categories.

Supported target platforms
- Windows, 32-bit and 64-bit
- macOS, 32-bit and 64-bit
- iOS, 32-bit and 64-bit
- Android, 32-bit and 64-bit
- Linux, 32-bit and 64-bit

General usability:
- Direct access to server data. Does not require installation of other data provider layers (such as BDE and ODBC)
- Interface compatible with standard data access methods, such as BDE and ADO
- VCL, LCL and FMX versions of library available
- Separated run-time and GUI specific parts allow you to create pure console applications such as CGI
- Unicode and national charset support

Network and connectivity:
- Does not require PostgreSQL client software and works directly through TCP/IP
- **Disconnected Model** with automatic connection control for working with data offline
- **Local Failover** for detecting connection loss and implicitly reexecuting certain operations
- Connection timeout and command timeout management
- **Secure connections** with SSL, SSH, and HTTP/HTTPS tunneling (using SecureBridge)
- Ability to search for installed PostgreSQL Server databases in a local network

Compatibility:
- **Full support of the latest versions of PostgreSQL**
- Support for all PostgreSQL data types
- **Compatible with all IDE versions starting with Delphi 6, C++Builder 6, Free Pascal**
- Includes provider for UniDAC Express Edition
- Wide reporting component support, including support for InfoPower, ReportBuilder, FastReport
- Support of all standard and third-party visual data-aware controls
- Allows you to use Professional Edition of Delphi and C++Builder to develop client/server applications

PostgreSQL Server technology support:
- Support for fast record insertion with the **TPgLoader component**
- Support for PostgreSQL Asynchronous Notification with the **TPgAlerter component**
- PostgreSQL sequences support
- Supports the possibility of retrieving last inserted OID value
- Advanced errors support
- Support for the PostgreSQL notices

PostgreSQL DataTypes:
- Support for PostgreSQL Protocol 2 and Protocol 3
- PostgreSQL Composite types support
- PostgreSQL domain types support
- Full support for the `DATE`, `TIME`, `TIMESTAMP`, and `INTERVAL` data types
- Advanced `LARGE OBJECT` support
- Advanced support of the `REFCURSOR` type
- Wrapper classes for geometric types support

**Performance:**
- High overall performance
- Fast controlled fetch of large data blocks
- Optimized string data storing
- Advanced connection pooling
- High performance of applying cached updates with `batches`
- Caching of calculated and lookup fields
- `Fast Locate` in a sorted DataSet
- Preparing of user-defined update statements

**Local data storage operations:**
- Database-independent data storage with `TVirtualTable` component
- `CachedUpdates` operation mode
- Local sorting and filtering, including by calculated and lookup fields
- Local master/detail relationship
- Master/detail relationship in CachedUpdates mode

**Data access and data management automation:**
- Automatic data updating with `TPgQuery`, `TPgTable`, and `TPgStoredProc` components
- Automatic record refreshing and locking
- Automatic query preparing
- Automatic checking for row modifications by another user
- Support for `ftWideMemo` field type in Delphi 2006 and higher

**Extended data access functionality:**
- Separate component for executing SQL statements
- Simplified access to table data with `TPgTable` component
- Ability to retrieve metadata information with `TPgMetaData` component
- BLOB compression support
- Support for using macros in SQL
- FmtBCD fields support
- Ability to customize update commands by attaching external components to `TPgUpdateSQL` objects
- Retrieval of output parameters from stored procedures and functions
- Automatic retrieval of default field values
- Deferred detail Dataset refresh in master/detail relationships
- MIDAS technology support

**Data exchange:**
- Transferring data between all types of TDataSet descendants with `TCRBatchMove` component
- Data export and import to/from XML (ADO format)
- Ability to synchronize positions in different DataSets
- Extended data management with `TPgDump` component

**Script execution:**
- Advanced script execution features with the `TPgScript` component
- Support for executing individual statements in scripts
- Support for executing huge scripts stored in files with dynamic loading
- Ability to break long-running query execution

**SQL execution monitoring:**
- Extended SQL tracing capabilities provided by `TPgSQLMonitor` component and `DBMonitor`
- Borland SQL Monitor support
- Ability to send messages to `DBMonitor` from any point in your program
- Ability to retrieve information about the last query execution

**Visual extensions:**
- Includes source code of enhanced TCRDBGrid data-aware grid control
- Customizable connection dialog
- Cursor changes during non-blocking execution

**Design-time enhancements:**
- `DataSet Manager` tool to control Dataset instances in the project
- Advanced design-time component and property editors
- Automatic design-time component linking
• More convenient data source setup with the TPgDataSource component
• Syntax highlighting in design-time editors

Resources:
• Code documentation and guides in the CHM, PDF, and HXS formats
• Many helpful demo projects

Licensing and support:
• Included annual PgDAC Subscription with Priority Support
• Licensed royalty-free per developer, per team, or per site

2.3 Requirements

PgDAC works directly through TCP/IP protocol and does not use the PostgreSQL client library.

2.4 Compatibility

PostgreSQL Compatibility
PgDAC supports PostgreSQL server versions from 8.0 to 12.

Microsoft Azure Database for PostgreSQL Compatibility
PgDAC supports Microsoft Azure Database for PostgreSQL.

Amazon RDS for PostgreSQL Compatibility
PgDAC supports Amazon RDS for PostgreSQL and Amazon Aurora.

Google Cloud for PostgreSQL Compatibility
PgDAC supports Google Cloud for PostgreSQL.

IDE Compatibility
PgDAC is compatible with the following IDEs:

- Embarcadero RAD Studio 10.4 Sydney (Requires Release 1)
  - Embarcadero Delphi 10.4 Sydney for Windows
  - Embarcadero Delphi 10.4 Sydney for macOS
  - Embarcadero Delphi 10.4 Sydney for Linux
  - Embarcadero Delphi 10.4 Sydney for iOS
  - Embarcadero Delphi 10.4 Sydney for Android
  - Embarcadero C++Builder 10.4 Sydney for Windows
  - Embarcadero C++Builder 10.4 Sydney for macOS
  - Embarcadero C++Builder 10.4 Sydney for Linux
  - Embarcadero C++Builder 10.4 Sydney for iOS
  - Embarcadero C++Builder 10.4 Sydney for Android

- Embarcadero RAD Studio 10.3 Rio (Requires Release 2 or Release 3)
  - Embarcadero Delphi 10.3 Rio for Windows
  - Embarcadero Delphi 10.3 Rio for macOS
  - Embarcadero Delphi 10.3 Rio for Linux
  - Embarcadero Delphi 10.3 Rio for iOS
  - Embarcadero Delphi 10.3 Rio for Android
  - Embarcadero C++Builder 10.3 Rio for Windows
  - Embarcadero C++Builder 10.3 Rio for macOS
  - Embarcadero C++Builder 10.3 Rio for Linux
  - Embarcadero C++Builder 10.3 Rio for iOS
  - Embarcadero C++Builder 10.3 Rio for Android

- Embarcadero RAD Studio 10.2 Tokyo
  - Embarcadero Delphi 10.2 Tokyo for Windows
  - Embarcadero Delphi 10.2 Tokyo for macOS
  - Embarcadero Delphi 10.2 Tokyo for Linux
  - Embarcadero Delphi 10.2 Tokyo for iOS
  - Embarcadero Delphi 10.2 Tokyo for Android
  - Embarcadero C++Builder 10.2 Tokyo for Windows
  - Embarcadero C++Builder 10.2 Tokyo for macOS
  - Embarcadero C++Builder 10.2 Tokyo for Linux
  - Embarcadero C++Builder 10.2 Tokyo for iOS
  - Embarcadero C++Builder 10.2 Tokyo for Android

- Embarcadero RAD Studio 10.1 Berlin
  - Embarcadero Delphi 10.1 Berlin for Windows
  - Embarcadero Delphi 10.1 Berlin for macOS
  - Embarcadero Delphi 10.1 Berlin for Linux
  - Embarcadero Delphi 10.1 Berlin for iOS
  - Embarcadero Delphi 10.1 Berlin for Android
  - Embarcadero C++Builder 10.1 Berlin for Windows
Embarcadero RAD Studio 10 Seattle
- Embarcadero Delphi 10 Seattle for Windows
- Embarcadero Delphi 10 Seattle for macOS
- Embarcadero Delphi 10 Seattle for iOS
- Embarcadero Delphi 10 Seattle for Android
- Embarcadero C++Builder 10 Seattle for Windows
- Embarcadero C++Builder 10 Seattle for macOS
- Embarcadero C++Builder 10 Seattle for iOS
- Embarcadero C++Builder 10 Seattle for Android

Embarcadero RAD Studio XE8
- Embarcadero Delphi XE8 for Windows
- Embarcadero Delphi XE8 for macOS
- Embarcadero Delphi XE8 for iOS
- Embarcadero Delphi XE8 for Android
- Embarcadero C++Builder XE8 for Windows
- Embarcadero C++Builder XE8 for macOS
- Embarcadero C++Builder XE8 for iOS
- Embarcadero C++Builder XE8 for Android

Embarcadero RAD Studio XE7
- Embarcadero Delphi XE7 for Windows
- Embarcadero Delphi XE7 for macOS
- Embarcadero Delphi XE7 for iOS
- Embarcadero Delphi XE7 for Android
- Embarcadero C++Builder XE7 for Windows
- Embarcadero C++Builder XE7 for macOS
- Embarcadero C++Builder XE7 for iOS
- Embarcadero C++Builder XE7 for Android

Embarcadero RAD Studio XE6
- Embarcadero Delphi XE6 for Windows
- Embarcadero Delphi XE6 for macOS
- Embarcadero Delphi XE6 for iOS
- Embarcadero Delphi XE6 for Android
- Embarcadero C++Builder XE6 for Windows
- Embarcadero C++Builder XE6 for macOS
o Embarcadero C++Builder XE6 for iOS
  o Embarcadero C++Builder XE6 for Android

• Embarcadero RAD Studio XE5 (Requires Update 2)
  o Embarcadero Delphi XE5 for Windows
  o Embarcadero Delphi XE5 for macOS
  o Embarcadero Delphi XE5 for iOS
  o Embarcadero Delphi XE5 for Android
  o Embarcadero C++Builder XE5 for Windows
  o Embarcadero C++Builder XE5 for macOS
  o Embarcadero C++Builder XE5 for iOS

• Embarcadero RAD Studio XE4
  o Embarcadero Delphi XE4 for Windows
  o Embarcadero Delphi XE4 for macOS
  o Embarcadero Delphi XE4 for iOS
  o Embarcadero C++Builder XE4 for Windows
  o Embarcadero C++Builder XE4 for macOS

• Embarcadero RAD Studio XE3 (Requires Update 2)
  o Embarcadero Delphi XE3 for Windows
  o Embarcadero Delphi XE3 for macOS
  o Embarcadero C++Builder XE3 for Windows
  o Embarcadero C++Builder XE3 for macOS

• Embarcadero RAD Studio XE2 (Requires Update 4 Hotfix 1)
  o Embarcadero Delphi XE2 for Windows
  o Embarcadero Delphi XE2 for macOS
  o Embarcadero C++Builder XE2 for Windows
  o Embarcadero C++Builder XE2 for macOS

• Embarcadero RAD Studio XE
  o Embarcadero Delphi XE
  o Embarcadero C++Builder XE

• Embarcadero RAD Studio 2010
  o Embarcadero Delphi 2010
  o Embarcadero C++Builder 2010

• CodeGear RAD Studio 2009 (Requires Update 3)
  o CodeGear Delphi 2009
  o CodeGear C++Builder 2009

• CodeGear RAD Studio 2007
- CodeGear Delphi 2007
- CodeGear C++Builder 2007
- Borland Developer Studio 2006
- Borland Delphi 2006
- Borland C++Builder 2006
- Borland Delphi 7
- Borland Delphi 6 (Requires Update Pack 2 – Delphi 6 Build 6.240)
- Borland C++Builder 6 (Requires Update Pack 4 – C++Builder 6 Build 10.166)
- Lazarus 2.0.10 and Free Pascal 3.2.0 for Windows, macOS, and Linux.

All the existing Delphi and C++Builder editions are supported: Architect, Enterprise, Professional, Community, and Starter.

Lazarus and Free Pascal are supported only in Trial Edition and Professional Edition with source code.

Supported Target Platforms
- Windows, 32-bit and 64-bit
- macOS, 32-bit and 64-bit
- Linux, 32-bit (only in Lazarus and Free Pascal) and 64-bit
- iOS, 32-bit and 64-bit
- Android, 32-bit and 64-bit

Note that support for 64-bit Windows and macOS was introduced in RAD Studio XE2, and is not available in older versions of RAD Studio. Support for iOS is available since RAD Studio XE4, but support for iOS 64-bit is available since RAD Studio XE8. Support for Android is available since RAD Studio XE5. Support for Linux 64-bit is available since RAD Studio 10.2 Tokyo. Support for macOS 64-bit is available since RAD Studio 10.3 Rio. Support for Android 64-bit is available since RAD Studio 10.3.3 Rio. Support for macOS 32-bit and iOS 32-bit was removed in RAD Studio 10.4.

Supported GUI Frameworks
- FireMonkey (FMX)
- Visual Component Library (VCL)
- Lazarus Component Library (LCL)
Devart Data Access Components Compatibility

All DAC products are compatible with each other.

But, to install several DAC products to the same IDE, it is necessary to make sure that all DAC products have the same common engine (BPL files) version. The latest versions of DAC products or versions with the same release date always have the same version of the common engine and can be installed to the same IDE.

© 1997-2021
Devar. All Rights Reserved.

2.5 Using Several DAC Products in One IDE

UniDAC, ODAC, SDAC, MyDAC, IBDAC, PgDAC, LiteDAC and VirtualDAC components use common base packages listed below:

Packages:
- dacXX.bpl
- dacvclXX.bpl
- dcldacXX.bpl

Note that product compatibility is provided for the current build only. In other words, if you upgrade one of the installed products, it may conflict with older builds of other products. In order to continue using the products simultaneously, you should upgrade all of them at the same time.

© 1997-2021
Devar. All Rights Reserved.

2.6 Component List

This topic presents a brief description of the components included in the PostgreSQL Data Access Components library. Click on the name of each component for more information. These components are added to the PgDAC page of the Component palette except for TCRBatchMove and TVirtualTable components. TCRBatchMove and TVirtualTable components are added to the Data Access page of the Component palette. Basic PgDAC components are included in all PgDAC editions. PgDAC Professional Edition components are not included in PgDAC Standard Edition.
### Basic PgDAC components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgConnection</td>
<td>Represents an open connection to a PostgreSQL database.</td>
</tr>
<tr>
<td>TPgQuery</td>
<td>Executes queries and operates record sets. It also provides flexible way to update data.</td>
</tr>
<tr>
<td>TPgSQL</td>
<td>Executes SQL statements and stored procedures, which do not return rowsets.</td>
</tr>
<tr>
<td>TPgTable</td>
<td>Lets you retrieve and update data in a single table without writing SQL statements.</td>
</tr>
<tr>
<td>TPgStoredProc</td>
<td>Has access to and executes stored procedures and functions.</td>
</tr>
<tr>
<td>TPgUpdateSQL</td>
<td>Lets you tune update operations for the DataSet component.</td>
</tr>
<tr>
<td>TPgDataSource</td>
<td>Provides an interface between PgDAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td>TPgScript</td>
<td>Executes sequences of SQL statements.</td>
</tr>
<tr>
<td>TPgSQLMonitor</td>
<td>Interface for monitoring dynamic SQL execution in PgDAC-based applications.</td>
</tr>
<tr>
<td>TPgConnectDialog</td>
<td>Used to build custom prompts for username, password and server name.</td>
</tr>
<tr>
<td>TVirtualTable</td>
<td>Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.</td>
</tr>
<tr>
<td>TVirtualDataSet</td>
<td>Dataset that processes arbitrary non-tabular data.</td>
</tr>
</tbody>
</table>

### PgDAC Professional Edition components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgEncryptor</td>
<td>Represents data encryption and decryption in client application.</td>
</tr>
<tr>
<td>TPgLoader</td>
<td>Provides quick loading of external data into the server database.</td>
</tr>
<tr>
<td>TPgDump</td>
<td>Serves to store a database or its parts as a script and also to restore database from received script.</td>
</tr>
</tbody>
</table>
2.7 Hierarchy Chart

Many PgDAC classes are inherited from standard VCL/LCL classes. The inheritance hierarchy chart for PgDAC is shown below. The PgDAC classes are represented by hyperlinks that point to their description in this documentation. A description of the standard classes can be found in the documentation of your IDE.

```
TObject
 |—TPersistent
 |—TComponent
 |   |—TCustomConnection
 |   |   |—TCustomDAConnection
 |   |   |—TPgConnection
 |   |—TDataSet
 |   |   |—TMemDataSet
 |   |   |   |—TCustomDADataset
 |   |   |   |—TCustomPgDataSet
 |   |   |   |—TPgQuery
 |   |   |—TCustomPgTable
 |   |   |—TPgTable
 |   |—TCustomPgStoredProc
 |   |   |—TPgStoredProc
 |   |—TDAMetaData
 |   |   |—TPgMetaData
 |   |   |—TVirtualTable
```
2.8 Editions

PostgreSQL Data Access Components comes in two editions: Standard and Professional.

The Standard edition includes the PgDAC basic connectivity components. PgDAC Standard Edition is a cost-effective solution for database application developers who are looking for high-performance connectivity to PostgreSQL for secure, reliable, and high-speed data transmission.

The Professional edition shows off the full power of PgDAC, enhancing PgDAC Standard Edition with support for PostgreSQL-specific functionality and advanced dataset management features.

You can get Source Access to PgDAC Professional Edition by purchasing a special PgDAC Professional Edition with Source Code, which includes the source code of all component classes. The source code of DataSet Manager and Migration Wizard is not distributed.

The matrix below compares features of PgDAC editions. See Features for the detailed list of PgDAC features.

PgDAC Edition Matrix

<table>
<thead>
<tr>
<th>Feature</th>
<th>Standard</th>
<th>Professional</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgTimeStamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPgDate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPgTime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPgInterval</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPgGeometric</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPgPoint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPgCircle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PgPointsArray</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPgLSeg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPgBox</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPgPath</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPgPolygon</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Direct connectivity**

| Connection without PostgreSQL client library |  |  |

**Desktop Application Development**

| Windows |  |  |
| macOS | × |  |
| Linux | × |  |

**Mobile Application Development**

| iOS | × |  |
| Android | × |  |

**Data Access Components**

| Base Components: |  |  |
| TPgConnection | | |
| TPgQuery | | |
| TPgSQL | ✓ |  |
| TPgTable | | |
| TPgStoredProc | | |
| TPgUpdateSQL | | |
| TPgDataSource | | |

| Script executing |  |  |
| TPgScript | × |  |

| Fast data loading into the server |  |  |
| TPgLoader | × |  |

**PostgreSQL Specific Components**

| Messaging between sessions and applications |  |  |
| TPgAlerter | × |  |

| Obtaining metadata about database objects |  |  |
| TPgMetaData | × |  |

| Storing a database as a script |  |  |
| TPgDump | × |  |
### Monitoring of per-component SQL execution

**TPgSQLMonitor**

| 3 | 3 | 3 |

### Additional components

#### Advanced connection dialog

**TPgConnectDialog**

| 3 | 3 | 3 |

#### Data encryption and decryption

**TPgEncryptor**

| X | 3 | 3 |

#### Data storing in memory table

**TVirtualTable**

| 3 | 3 | 3 |

#### Dataset that wraps arbitrary non-tabular data

**TVirtualDataSet**

| 3 | 3 | 3 |

#### Advanced DBGrid with extended functionality

**TCRDBGrid**

| X | 3 | 3 |

#### Records transferring between datasets

**TCRBatchMove**

| X | 3 | 3 |

### Design-Time Features

#### Enhanced component and property editors

| 3 | 3 | 3 |

#### DataSet Manager

| X | 3 | 3 |

### Cross IDE Support

#### Lazarus and Free Pascal Support

| X | SRC | 3 |

---

1 Available only in Professional Edition with Source Code.

**© 1997-2021 Devart. All Rights Reserved.**

**2.9 Licensing**

PLEASE READ THIS LICENSE AGREEMENT CAREFULLY. BY INSTALLING OR USING THIS SOFTWARE, YOU INDICATE ACCEPTANCE OF AND AGREE TO BECOME BOUND BY THE TERMS AND CONDITIONS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, DO NOT INSTALL OR USE THIS SOFTWARE AND PROMPTLY RETURN IT TO DEVART.

**INTRODUCTION**

© 2021 Devart
This Devart end-user license agreement ("Agreement") is a legal agreement between you (either an individual person or a single legal entity) and Devart, for the use of PgDAC software application, source code, demos, intermediate files, printed materials, and online or electronic documentation contained in this installation file. For the purpose of this Agreement, the software program(s) and supporting documentation will be referred to as the "Software".

LICENSE

1. GRANT OF LICENSE

The enclosed Software is licensed, not sold. You have the following rights and privileges, subject to all limitations, restrictions, and policies specified in this Agreement.

1.1. If you are a legally licensed user, depending on the license type specified in the registration letter you have received from Devart upon purchase of the Software, you are entitled to either:

- install and use the Software on one or more computers, provided it is used by 1 (one) for the sole purposes of developing, testing, and deploying applications in accordance with this Agreement (the "Single Developer License"); or
- install and use the Software on one or more computers, provided it is used by up to 4 (four) developers within a single company at one physical address for the sole purposes of developing, testing, and deploying applications in accordance with this Agreement (the "Team Developer License"); or
- install and use the Software on one or more computers, provided it is used by developers in a single company at one physical address for the sole purposes of developing, testing, and deploying applications in accordance with this Agreement (the "Site License").

1.2. If you are a legally licensed user of the Software, you are also entitled to:

- make one copy of the Software for archival purposes only, or copy the Software onto the hard disk of your computer and retain the original for archival purposes;
- develop and test applications with the Software, subject to the Limitations below;
- create libraries, components, and frameworks derived from the Software for personal use only;
- deploy and register run-time libraries and packages of the Software, subject to the Redistribution policy defined below.

1.3. You are allowed to use evaluation versions of the Software as specified in the Evaluation section.

No other rights or privileges are granted in this Agreement.
2. LIMITATIONS

Only legally registered users are licensed to use the Software, subject to all of the conditions of this Agreement. Usage of the Software is subject to the following restrictions.

2.1. You may not reverse engineer, decompile, or disassemble the Software.

2.2. You may not build any other components through inheritance for public distribution or commercial sale.

2.3. You may not use any part of the source code of the Software (original or modified) to build any other components for public distribution or commercial sale.

2.4. You may not reproduce or distribute any Software documentation without express written permission from Devart.

2.5. You may not distribute and sell any portion of the Software without integrating it into your Applications as Executable Code, except a Trial version that can be distributed for free as original Devart's PgDAC Trial package.

2.6. You may not transfer, assign, or modify the Software in whole or in part. In particular, the Software license is non-transferable, and you may not transfer the Software installation package.

2.7. You may not remove or alter any Devart's copyright, trademark, or other proprietary rights notice contained in any portion of Devart units, source code, or other files that bear such a notice.

3. REDISTRIBUTION

The license grants you a non-exclusive right to compile, reproduce, and distribute any new software programs created using PgDAC. You can distribute PgDAC only in compiled Executable Programs or Dynamic-Link Libraries with required run-time libraries and packages.

All Devart's units, source code, and other files remain Devart's exclusive property.

4. TRANSFER

You may not transfer the Software to any individual or entity without express written permission from Devart. In particular, you may not share copies of the Software under “Single Developer License” and “Team License” with other co-developers without obtaining proper license of these copies for each individual.
5. TERMINATION
Devart may immediately terminate this Agreement without notice or judicial resolution in the event of any failure to comply with any provision of this Agreement. Upon such termination you must destroy the Software, all accompanying written materials, and all copies.

6. EVALUATION
Devart may provide evaluation ("Trial") versions of the Software. You may transfer or distribute Trial versions of the Software as an original installation package only. If the Software you have obtained is marked as a "Trial" version, you may install and use the Software for a period of up to 60 calendar days from the date of installation (the "Trial Period"), subject to the additional restriction that it is used solely for evaluation of the Software and not in conjunction with the development or deployment of any application in production. You may not use applications developed using Trial versions of the Software for any commercial purposes. Upon expiration of the Trial Period, the Software must be uninstalled, all its copies and all accompanying written materials must be destroyed.

7. WARRANTY
The Software and documentation are provided "AS IS" without warranty of any kind. Devart makes no warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or use.

8. SUBSCRIPTION AND SUPPORT
The Software is sold on a subscription basis. The Software subscription entitles you to download improvements and enhancement from Devart's web site as they become available, during the active subscription period. The initial subscription period is one year from the date of purchase of the license. The subscription is automatically activated upon purchase, and may be subsequently renewed by Devart, subject to receipt applicable fees. Licensed users of the Software with an active subscription may request technical assistance with using the Software over email from the Software development. Devart shall use its reasonable endeavours to answer queries raised, but does not guarantee that your queries or problems will be fixed or solved.

Devart reserves the right to cease offering and providing support for legacy IDE versions.

9. COPYRIGHT
The Software is confidential and proprietary copyrighted work of Devart and is protected by international copyright laws and treaty provisions. You may not remove the copyright notice
2.10 Getting Support

This page lists several ways you can find help with using PgDAC and describes the PgDAC Priority Support program.

Support Options

There are a number of resources for finding help on installing and using PgDAC.

- You can find out more about PgDAC installation or licensing by consulting the Licensing section.
- You can get community assistance and technical support on the PgDAC Community Forum.
- You can get advanced technical assistance by PgDAC developers through the PgDAC Priority Support program.

If you have a question about ordering PgDAC or any other Devart product, please contact sales@devart.com.

PgDAC Priority Support

PgDAC Priority Support is an advanced product support service for getting expedited individual assistance with PgDAC-related questions from the PgDAC developers themselves. Priority Support is carried out over email and has two business days response policy. Priority Support is available for users with an active PgDAC Subscription.

To get help through the PgDAC Priority Support program, please send an email to support@devart.com describing the problem you are having. Make sure to include the following information in your message:

- The version of Delphi or C++Builder you are using.
- Your PgDAC Registration number.
- Full PgDAC edition name and version number. You can find both of these from the PgDAC | PgDAC About menu in the IDE.
2.11 FAQ

This page contains a list of Frequently Asked Questions for PostgreSQL Data Access Components.

If you have encountered a question with using PgDAC, please browse through this list first. If this page does not answer your question, refer to the Getting Support topic in PgDAC help.

Installation and Deployment

1. I am having a problem installing PgDAC or compiling PgDAC-based projects...

You may be having a compatibility issue that shows up in one or more of the following forms:
- Get a "Setup has detected already installed DAC packages which are incompatible with current version" message during PgDAC installation.
- Get a "Procedure entry point ... not found in ..." message when starting IDE.
- Get a "Unit ... was compiled with a different version of ..." message on compilation.

You can have such problems if you installed incompatible PgDAC, SDAC, ODAC or IBDAC versions. All these products use common base packages. The easiest way to avoid the problem is to uninstall all installed DAC products and then download from our site and install the last builds.

2. What software should be installed on a client computer for PgDAC-based applications to work?

Usually, you do not need any additional files. The only exceptions to this rule are listed below:
- If you are connecting in Client mode, (TPgConnection.Options.Direct = False), you need PostgreSQL client library.
- If you are using SSL (TPgConnection.Options.Protocol = mpSSL), you need the OpenSSL library files - ssleay32.dll and libeay32.dll.

3. When I try to connect to the server, I get an error "PostgreSQL client library
couldn't be loaded."

You are using TPgConnection.Options.Direct := False mode and the client library is not available for your application.

Windows: You should copy client file PostgreSQL client library to a folder available to the executable unit of your program. For example, to the folder containing the executable or to the Windows system folder. For more details, see the description of LoadLibrary and the PATH environment variable.

Linux: You should copy the client file libPostgreSQLclient.so.X to the folder available to the executable unit of your program. For more details, see the description of the dlopen function and the LD_LIBRARY_PATH environment variable.

Licensing and Subscriptions
1. Am I entitled to distribute applications written with PgDAC?

If you have purchased a full version of PgDAC, you are entitled to distribute pre-compiled programs created with its use. You are not entitled to propagate any components inherited from PgDAC or using PgDAC source code. For more information see the License.rtf file in your PgDAC installation directory.

2. Can I create components using PgDAC?

You can create your own components that are inherited from PgDAC or that use the PgDAC source code. You are entitled to sell and distribute compiled application executables that use such components, but not their source code and not the components themselves.

3. I have a registered version of PgDAC. Will I need to pay to upgrade to future versions?

All upgrades to future versions are free to users with an active PgDAC Subscription.

4. What are the benefits of the PgDAC Subscription Program?

The PgDAC Subscription Program is an annual maintenance and support service for PgDAC users.

Users with a valid PgDAC Subscription get the following benefits:
  o Access to new versions of PgDAC when they are released
  o Access to all PgDAC updates and bug fixes
  o Product support through the PgDAC Priority Support program
  o Notification of new product versions

© 2021 Devart
**Priority Support** is an advanced product support program which offers you expedited individual assistance with PgDAC-related questions from the PgDAC developers themselves. Priority Support is carried out over email and has a two business day response policy.

**5. Can I use my version of PgDAC after my Subscription expires?**

Yes, you can. PgDAC version licenses are perpetual.

**6. I want a PgDAC Subscription! How can I get one?**

An annual PgDAC Subscription is included when ordering or upgrading to any registered (non-Trial) edition of PgDAC.

You can renew your PgDAC Subscription on the [PgDAC Ordering Page](#). For more information, please contact sales@devart.com.

**7. How do I upgrade?**

To upgrade to new PgDAC versions, you can get a Version Update from the [PgDAC Ordering Page](#). For more information, please contact sales@devart.com.

**Performance**

**1. How productive is PgDAC?**

PgDAC uses a low-level protocol to access the database server. This allows PgDAC to achieve high performance. From time to time we compare PgDAC with other products, and PgDAC always takes first place.

**2. Why does the Locate function work so slowly the first time I use it?**

Locate is performed on the client. So if you had set FetchAll to False when opening your dataset, cached only some of the rows on the client, and then invoked Locate, PgDAC will have to fetch all the remaining rows from the server before performing the operation. On subsequent calls, Locate should work much faster.

If the Locate method keeps working slowly on subsequent calls or you are working with FetchAll=True, try the following. Perform local sorting by a field that is used in the Locate method. Just assign corresponding field name to the IndexFieldNames property.

**How To**

**1. How can I determine which version of PgDAC I am using?**

You can determine your PgDAC version number in several ways:

- During installation of PgDAC, consult the PgDAC Installer screen.
- After installation, see the `history.html` file in your PgDAC installation directory.
At design-time, select PostgreSQL | About PgDAC from the main menu of your IDE.

At run-time, check the value of the PgDACVersion and DACVersion constants.

2. How can I stop the cursor from changing to an hour glass during query execution?

Just set the DBAccess.ChangeCursor variable to False anywhere in your program. The cursor will stop changing after this command is executed.

3. How can I execute a query saved in the SQLInsert, SQLUpdate, SQLDelete, or SQLRefresh properties of a PgDAC dataset?

The values of these properties are templates for query statements, and they cannot be manually executed. Usually there is no need to fill these properties because the text of the query is generated automatically.

In special cases, you can set these properties to perform more complicated processing during a query. These properties are automatically processed by PgDAC during the execution of the Post, Delete, or RefreshRecord methods, and are used to construct the query to the server. Their values can contain parameters with names of fields in the underlying data source, which will be later replaced by appropriate data values.

For example, you can use the SQLInsert template to insert a row into a query instance as follows.

- Fill the SQLInsert property with the parameterized query template you want to use.
- Call Insert.
- Initialize field values of the row to insert.
- Call Post.

The value of the SQLInsert property will then be used by PgDAC to perform the last step.

Setting these properties is optional and allows you to automatically execute additional SQL statements, add calls to stored procedures and functions, check input parameters, and/or store comments during query execution. If these properties are not set, the PgDAC dataset object will generate the query itself using the appropriate insert, update, delete, or refresh record syntax.

4. How can I get a list of the databases on the server?

Use the TPgConnection.GetDatabaseNames method.

5. How can I get a list of the tables list in a database?

Use the TPgConnection.GetTableNames method.
6. Some questions about the visual part of PgDAC

The following situations usually arise from the same problem:
  o I set the Debug property to True but nothing happens!
  o While executing a query, the screen cursor does not change to an hour-glass.
  o Even if I have LoginPromp set to True, the connect dialog does not appear.

To fix this, you should add the PgDACVcl (for Windows) or PgDACClx (for Linux) unit to the uses clause of your project.

General Questions
1. I would like to develop an application that works with PostgreSQL Server. Should I use PgDAC or DbxMda?

DbxMda is our dbExpress driver for PostgreSQL. dbExpress technology serves for providing a more or less uniform way to access different servers (SQL Server, PostgreSQL, Oracle and so on). It is based on drivers that include server-specific features. Like any universal tool, in many specialized cases dbExpress providers lose some functionality. For example, the dbExpress design-time is quite poor and cannot be expanded.

PgDAC is a specialized set of components for PostgreSQL, which has advanced server-specific design-time and a component interface similar to that of BDE.

We tried to include maximal support of PostgreSQL-specific features in both DbxMda and PgDAC. However, the nature of dbExpress technology has some insurmountable restrictions. For example, Unicode fields cannot be passed from a driver to dbExpress.

PgDAC and DbxMda use the same kernel and thus have similar performance. In some cases dbExpress is slower because data undergoes additional conversion to correspond to dbExpress standards.

To summarise, if it is important for you to be able to quickly adapt your application to a database server other than PostgreSQL, it is probably better to use DbxMda. In other cases, especially when migrating from BDE or ADO, you should use PgDAC.

2. Are the PgDAC connection components thread-safe?

Yes, PgDAC is thread-safe but there is a restriction. The same TPgConnection object cannot be used in several threads. So if you have a multithreaded application, you should have a TPgConnection object for each thread that uses PgDAC.

3. Behaviour of my application has changed when I upgraded PgDAC. How can I restore the old behaviour with the new version?
We always try to keep PgDAC compatible with previous versions, but sometimes we have to change behaviour of PgDAC in order to enhance its functionality, or avoid bugs. If either of changes is undesirable for your application, and you want to save the old behaviour, please refer to the "Compatibility with previous versions" topic in PgDAC help. This topic describes such changes, and how to revert to the old PgDAC behaviour.

4. When editing a DataSet, I get an exception with the message 'Update failed. Found %d records.' or 'Refresh failed. Found %d records.'

This error occurs when the database server is unable to determine which record to modify or delete. In other words, there are either more than one record or no records that suit the UPDATE criteria. Such situation can happen when you omit the unique field in a SELECT statement (TCustomDADataSet.SQL) or when another user modifies the table simultaneously. This exception can be suppressed. Refer to TCustomPgDataSet.Options.StrictUpdate topic in PgDAC help for more information.

5. I have problems using BIGINT and INT UNSIGNED fields as key fields in master/detail relationships, and accessing values of such fields through the Field.Value property.

Fields of this type are represented in Delphi by TLargeIntField objects. In some versions of Delphi, you cannot access these fields through the Value property (for more information see the SetVarValue protected method of TLargeintField in the DB unit). To avoid this problem, you can change the field type to INT, which is usually sufficient for key fields. Alternatively, you can avoid using Value.

For master/detail relationships the problem can be avoided only by changing type of the key field to INT, as Delphi's master/detail mechanism works through Field.Value.

6. On accessing server I get a 'PostgreSQL server has gone away' or 'Lost connection to PostgreSQL server during query' error.

First of all, you should find out what causes the problem. The list of most frequent reasons for this error to occur is below.

- **Client side:** The value of TPgConnection.ConnectionTimeout or TCustomPgDataSet.CommandTimeout is too small. To check this hypothesis, try setting TCustomPgDataSet.CommandTimeout to 0 (infinitive) and TPgConnection.ConnectionTimeout to 300.
- **Server side:** PostgreSQL server has closed the connection. Almost always it is because the value of wait_timeout variable is too small. Try increasing it. If this solution is not possible (for example, because you don't have enough rights), you should invoke
PgConnection.Ping with an interval less than wait_timeout. Use TTimer in TPgConnection thread to accomplish this task.

- Unstable connection (GPRS etc.). In case of unstable connection you can adapt PgDAC to work in such conditions by changing some of its settings. For more information please see the “Working in Unstable Networks” article in the PgDAC help documentation.

If the connection is lost, PgDAC tries to reconnect to server. However, your last command will probably not be executed, and you should repeat it again. PgDAC does not try to reconnect if a transaction has started or if at least one of statements is prepared.

7. Some problems using TCustomDADataSet.FetchAll=False mode

The following problems may appear when using FetchAll=False mode:

- I have problems working with temporary tables.
- I have problems working with transactions.
- Sometimes my application hangs on applying changes to the database.

Usage of FetchAll=False mode has many advantages; however, it also has some restrictions since it requires an additional connection to server for data fetching to be created. The additional connection is created to prevent the main connection from blocking.

These problems can be avoided by setting the FetchAll property. Please see description of the FetchAll property and the CreateConnection option in PgDAC help for more information.

8. I get an error when opening a Stored Procedure that returns a result set.

Probably this is a bug of the PostgreSQL Server protocol with prepared stored procedures that return record sets. It occurs in the following cases:

- After a call to the Prepare method of PgStoredProc, if the latter had already prepared and opened. The following piece of code demonstrates the problem:

```pascal
PgStoredProc.Prepare;
PgStoredProc.Open;
PgStoredProc.UnPrepare;
PgStoredProc.Prepare;
```

- After a call to the PgStoredProc.Execute method, if the stored procedure returns more than one record set.
3 Getting Started

This page contains a quick introduction to setting up and using the PostgreSQL Data Access Components library. It gives a walkthrough for each part of the PgDAC usage process and points out the most relevant related topics in the documentation.

- What is PgDAC?
- Installing PgDAC.
- Working with the PgDAC demo projects.
- Compiling and deploying your PgDAC project.
- Using the PgDAC documentation.
- How to get help with PgDAC.

What is PgDAC?

PostgreSQL Data Access Components (PgDAC) is a component library which provides direct connectivity to PostgreSQL for Delphi, C++Builder and Lazarus (FPC), and helps you develop fast PostgreSQL-based database applications with these environments.

Many PgDAC classes are based on VCL, LCL and FMX classes and interfaces. PgDAC is a replacement for the Borland Database Engine, it provides native database connectivity, and is specifically designed as an interface to the PostgreSQL database.

An introduction to PgDAC is provided in the Overview section.

A list of the PgDAC features you may find useful is listed in the Features section.

An overview of the PgDAC component classes is provided in the Components List section.

Installing PgDAC

To install PgDAC, complete the following steps.

1. Choose and download the version of the PgDAC installation program that is compatible with your IDE. For instance, if you are installing PgDAC 1.00, you should use the following files:
   - For BDS 2006 and Turbo - pdac100d10*.exe
   - For Delphi 7 - pdac1100d7*.exe

   For more information, visit the PgDAC download page.

2. Close all running Borland applications.

3. Launch the PgDAC installation program you downloaded in the first step and follow the
instructions to install PgDAC.

By default, the PgDAC installation program should install compiled PgDAC libraries automatically on all IDEs.

To check if PgDAC has been installed properly, launch your IDE and make sure that the PgDAC page has been added to the Component palette and that a PgDAC menu was added to the Menu bar.

If you have bought PgDAC Professional Edition with Source Code, you will be able to download both the compiled version of PgDAC and the PgDAC source code. The installation process for the compiled version is standard, as described above. The PgDAC source code must be compiled and installed manually. Consult the supplied ReadmeSrc.html file for more details.

To find out what gets installed with PgDAC or to troubleshoot your PgDAC installation, visit the Installation topic.

Working with the PgDAC demo projects

The PgDAC installation package includes a number of demo projects that demonstrate PgDAC capabilities and use patterns. The PgDAC demo projects are automatically installed in the PgDAC installation folder.

To quickly get started working with PgDAC, launch and explore the introductory PgDAC demo project, PgDacDemo, from your IDE. This demo project is a collection of demos that show how PgDAC can be used. The project creates a form which contains an explorer panel for browsing the included demos and a view panel for launching and viewing the selected demo.

**PgDACDemo Walkthrough**

1. Launch your IDE.
2. Choose File | Open Project from the menu bar.
3. Find the PgDAC directory and open the PgDacDemo project. This project should be located in the Demos\PgDacDemo folder.

For example, if you are using Borland Developer Studio 2006, the demo project may be found at

\Program Files\Devart\PgDac for Delphi 2006\Demos\Win32\PgDacDemo\PgDacDemo.bdsproj

4. Select Run | Run or press F9 to compile and launch the demo project. PgDacDemo should start, and a full-screen PgDAC Demo window with a toolbar, an explorer panel, and a view panel will open. The explorer panel will contain the list of all demo sub-projects included in
*PgDACDemo*, and the view panel will contain an overview of each included demo. At this point, you will be able to browse through the available demos, read their descriptions, view their source code, and see the functionality provided by each demo for interacting with PostgreSQL. However, you will not be able to actually retrieve data from PostgreSQL or execute commands until you connect to the database.

5. Click on the "Connect" button on the *PgDacDemo* toolbar. A Connect dialog box will open. Enter the connection parameters you use to connect to your PostgreSQL server and click "Connect" in the dialog box.

Now you have a fully functional interface to your PostgreSQL server. You will be able to go through the different demos, to browse tables, create and drop objects, and execute SQL commands.

**Warning!** All changes you make to the database you are connected to, including creating and dropping objects used by the demo, will be permanent. Make sure you specify a test database in the connection step.

6. Click on the "Create" button to create all objects that will be used by *PgDacDemo*. If some of these objects already exist in the database you have connected to, the following error message will appear.

   **An error has occurred:**
   
   #42S01Table 'dept' already exists
   
   You can manually create objects required for demo by using the following file: %PgDAC%
   
   %PgDAC% is the PgDAC installation path on your computer.

   **Ignore this exception?**

   This is a standard warning from the object execution script. Click "Yes to All" to ignore this message. *PgDacDemo* will create the *PgDacDemo* objects on the server you have connected to.

7. Choose a demo that demonstrates an aspect of working with PostgreSQL that you are interested in, and play with the demo frame in the view window on the right. For example, to find out more about how to work with PostgreSQL tables, select the Table demo from the "Working with Components" folder. A simple PostgreSQL table browser will open in the view panel which will let you open a table in your database by specifying its name and clicking on the Open button.

8. Click on the "Demo source" button in the *PgDacDemo* toolbar to find out how the demo you selected was implemented. The source code behind the demo project will appear in the view panel. Try to find the places where PgDAC components are used to connect to the database.
9. Click on the "Form as text" button in the PgDacDemo toolbar to view the code behind the interface to the demo. Try to find the places where PgDAC components are created on the demo form.

10. Repeat these steps for other demos listed in the explorer window. The available demos are organized in three folders.

Working with components

A collection of projects that show how to work with basic PgDAC components.

General demos

A collection of projects that show off the PgDAC technology and demonstrate some ways of working with data.

PostgreSQL-specific demos

A collection of projects that demonstrate how to incorporate PostgreSQL features in database applications.

11. When you are finished working with the project, click on the "Drop" button in the PgDacDemo toolbar to remove all schema objects added in Step 6.

Other PgDAC demo projects

PgDAC is accompanied by a number of other demo projects. A description of all PgDAC demos is located in the Demo Projects topic.

Compiling and deploying your PgDAC project

Compiling PgDAC-based projects

By default, to compile a project that uses PgDAC classes, your IDE compiler needs to have access to the PgDAC dcu (obj) files. If you are compiling with runtime packages, the compiler will also need to have access to the PgDAC bpl files. All the appropriate settings for both these scenarios should take place automatically during installation of PgDAC. You should only need to modify your environment manually if you are using one of the PgDAC editions that comes with source code - PgDAC Professional Edition with Source Code or PgDAC Developer Edition with Source Code.

You can check that your environment is properly configured by trying to compile one of the PgDAC demo projects. If you have no problems compiling and launching the PgDAC demos, your environment has been properly configured.

For more information about which library files and environment changes are needed for
compiling PgDAC-based projects, consult the Installation topic.

Deploying PgDAC-based projects

To deploy an application that uses PgDAC, you will need to make sure the target workstation has access to the following files.

- The PostgreSQL client library, if connecting using PostgreSQL client.
- The PgDAC bpl files, if compiling with runtime packages.

If you are evaluating deploying projects with PgDAC Trial Edition, you will also need to deploy some additional bpl files with your application even if you are compiling without runtime packages. As another trial limitation for C++Builder, applications written with PgDAC Trial Edition for C++Builder will only work if the C++Builder IDE is launched. More information about PgDAC Trial Edition limitations is provided here.

A list of the files which may need to be deployed with PgDAC-based applications is included in the Deployment topic.

Using the PgDAC documentation

The PgDAC documentation describes how to install and configure PgDAC, how to use PgDAC Demo Projects, and how to use the PgDAC libraries.

The PgDAC documentation includes a detailed reference of all PgDAC components and classes. Many of the PgDAC components and classes inherit or implement members from other VCL, LCL and FMX classes and interfaces. The product documentation also includes a summary of all members within each of these classes. To view a detailed description of a particular component, look it up in the Components List section. To find out more about a specific standard VCL/LCL class a PgDAC component is inherited from, see the corresponding topic in your IDE documentation.

At install time, the PgDAC documentation is integrated into your IDE. It can be invoked from the PgDAC menu added to the Menu Bar, or by pressing F1 in an object inspector or on a selected code segment.

How to get help with PgDAC

There are a number of resources for finding help on using PgDAC classes in your project.

- If you have a question about PgDAC installation or licensing, consult the Licensing section.
- You can get community assistance and PgDAC technical support on the PgDAC Support Forum.
3.1 Installation

This topic contains the environment changes made by the PgDAC installer. If you are having problems using PgDAC or compiling PgDAC-based products, check this list to make sure your system is properly configured.

Compiled versions of PgDAC are installed automatically by PgDAC Installer for all supported IDEs except for Lazarus. Version of PgDAC with Source Code must be installed manually. Installation of PgDAC from sources is described in the supplied ReadmeSrc.html file.

Before installing PgDAC ...

Two versions of PgDAC cannot be installed in parallel for the same IDE, and, since the Devart Data Access Components products have some shared bpl files, newer versions of PgDAC may be incompatible with older versions of ODAC, IBDAC, SDAC, and UniDAC.

So before installing a new version of PgDAC, uninstall all previous versions of PgDAC you may have, and check if your new install is compatible with other Devart Data Access Components products you have installed. For more information please see Using several products in one IDE. If you run into problems or have any compatibility questions, please email pgdac@devart.com

Note: You can avoid performing PgDAC uninstallation manually when upgrading to a new version by directing the PgDAC installation program to overwrite previous versions. To do this, execute the installation program from the command line with a / f or ce parameter (Start | Run and type pgdacXX. exe / f or ce, specifying the full path to the appropriate version of the installation program).

Installed packages
Delphi/C++Builder Win32 project packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>dacXX.bpl</td>
<td>DAC run-time package</td>
<td>Windows\System32</td>
</tr>
<tr>
<td>dcldacXX.bpl</td>
<td>DAC design-time package</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>dacvclXX.bpl*</td>
<td>DAC VCL support package</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>pgdacXX.bpl</td>
<td>PgDAC run-time package</td>
<td>Windows\System32</td>
</tr>
<tr>
<td>dclpgdacXX.bpl</td>
<td>PgDAC design-time package</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>dclpgsqlmonXX.bpl</td>
<td>TPgSQLMonitor component</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>pgdacvclXX.bpl*</td>
<td>VCL support package</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>crcontrolsXX.bpl</td>
<td>TCRDBGrid component</td>
<td>Delphi\Bin</td>
</tr>
</tbody>
</table>

Additional packages for using PgDAC managers and wizards

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>datasetmanagerXX.bpl</td>
<td>DataSet Manager package</td>
<td>Delphi\Bin</td>
</tr>
</tbody>
</table>

Environment Changes

To compile PgDAC-based applications, your environment must be configured to have access to the PgDAC libraries. Environment changes are IDE-dependent.

For all instructions, replace %PgDAC% with the path to your PgDAC installation directory

Delphi

- %PgDAC% Lib should be included in the Library Path accessible from Tools | Environment options | Library.

The PgDAC Installer performs Delphi environment changes automatically for compiled versions of PgDAC.

C++Builder

C++Builder 6:
• \$(BCB)\ PgDAC\ Lib should be included in the Library Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.

• \$(BCB)\ PgDAC\ Include should be included in the Include Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.

C++Builder 2006, 2007:
• \$(BCB)\ PgDAC\ Lib should be included in the Library search path of the Default Project Options accessible from Project | Default Options | C++Builder | Linker | Paths and Defines.

• \$(BCB)\ PgDAC\ Include should be included in the Include search path of the Default Project Options accessible from Project | Default Options | C++Builder | C++ Compiler | Paths and Defines.

The PgDAC Installer performs C++Builder environment changes automatically for compiled versions of PgDAC.

Lazarus

The PgDAC installation program only copies PgDAC files. You need to install PgDAC packages to the Lazarus IDE manually. Open %PgDAC%\Source\Lazarus1\dclpgdac10.lpk (for Trial version %PgDAC%\Packages\dclpgdac10.lpk) file in Lazarus and press the Install button. After that Lazarus IDE will be rebuilt with PgDAC packages.

Do not press the the Compile button for the package. Compiling will fail because there are no PgDAC sources.

To check that your environment has been properly configured, try to compile one of the demo projects included with PgDAC. The PgDAC demo projects are located in %PgDAC%/Demos.

DBMonitor

DBMonitor is an easy-to-use tool to provide visual monitoring of your database applications. It is provided as an alternative to Borland SQL Monitor which is also supported by PgDAC. DBMonitor is intended to hamper application being monitored as little as possible. For more information, visit the DBMonitor page online.

3.2 Connecting to PostgreSQL

This tutorial describes how to connect to PostgreSQL using the TPgConnection component.

1. Requirements
2. General Information
3. Creating a Connection
   o 3.1 Connecting at Design-Time
      ▪ 3.1.1 Using TPgConnection Editor
      ▪ 3.1.2 Using Object Inspector
   o 3.2 Connecting at Runtime
4. Opening a Connection
5. Modifying a Connection
6. Closing a Connection
7. Additional Information
8. See Also

Requirements

This tutorial assumes that you have installed PgDAC and run the database server and the IDE. You need to know the server address, the port number (if you use a port other than the default port 5432), the database name, the schema, and the username and password. To connect at runtime, add the PgAccess unit to the uses clause for Delphi or include the PgAccess.hpp header file for C++ Builder.

General Information

To establish a connection to the server, set up the properties of the TPgConnection component: Server, Port, Database, Schema, Username, and Password. You can also specify all connection parameters in the ConnectString property.

Creating a Connection

Connecting at Design-Time

The following assumes that you have already created or opened an existing form in the IDE. At design-time, you can set up a TPgConnection object in the TPgConnection Editor or Object Inspector.

1. Find the TPgConnection component in the PgDAC category on the Tool Palette.
2. Double-click the component. A new object will appear on the form. If this is the first object TPgConnection in this unit, it will be named PgConnection1.

Using TPgConnection Editor

1. Double-click the PgConnection1 object.
2. Specify the DNS name or IP address of the PostgreSQL server in the Server edit box.
3. If you use a port other than the default port 5432, specify it in the Port edit box.
4. Specify the username (postgres by default) in the Username edit box.
5. Specify the password (postgres by default) in the Password edit box.
6. Specify the database name in the Database edit box.
7. Specify the schema (public by default) in the Schema edit box.

Using Object Inspector
1. Select the PgConnection1 object on the form.
2. Set the Database property to the database name.
3. Set the Password property to the password (postgres by default).
4. If you use a port other than the default port 5432, set the Port property to the port.
5. Set Schema property to the database schema (public by default).
6. Set the Server property to the DNS name or IP address of the PostgreSQL server.
7. Set the Username property to the username (postgres by default).

Connecting at Runtime
The same connection parameters at runtime are set up as follows:

**Delphi**

```delphi
var
  PgConnection1: TPgConnection;
begin
  PgConnection1 := TPgConnection.Create(nil);
  try
    // adds connection parameters
    PgConnection1.Server := 'server';
    PgConnection1.Database := 'database';
    PgConnection1.Username := 'username';
    PgConnection1.Password := 'password';
    PgConnection1.Port := 5432;
    // disables a login prompt
    PgConnection1.LoginPrompt := False;
    // opens a connection
    PgConnection1.Open;
  finally
    PgConnection1.Free;
  end;
end;
```

**C++ Builder**

```cpp
TPgConnection* PgConnection1 = new TPgConnection(NULL);
try {
  // adds connection parameters
  PgConnection1->Server = "server";
  PgConnection1->Database = "database";
  PgConnection1->Username = "username";
```
Opening a Connection

To open a connection at run-time, call the `Open` method:

**Delphi**

```delphi
PgConnection1.Open;
```

**C++Builder**

```cpp
PgConnection1->Open;
```

Another way to open a connection at runtime is to set the `Connected` property to `True`:

**Delphi**

```delphi
PgConnection1.Connected := True;
```

**C++Builder**

```cpp
PgConnection1->Connected = True;
```

You can also set up the `Connected` property at design-time in the Object Inspector.

Modifying a Connection

You can modify a connection by changing properties of the `TPgConnection` object. Note that while some of the object's properties can be altered without changing the state of a connection, in most cases, a connection is closed when a new value is assigned to the property. For example, if you change the value of the `Server` property, a connection is closed immediately and you need to reopen it manually.

Closing a Connection

To close a connection, call the `Close` method or set the `Connected` property to `False`:

**Delphi**

```delphi
PgConnection1.Close;
```

or:

```cpp
PgConnection1->Free();
```
3.3 Deleting Data From Tables

This tutorial describes how to delete data from a table using the TPgQuery and TPgTable components.

1. Requirements
2. General Information
3. Using the DataSet Functionality
4. **Building DML Statements Manually**
   - 4.1 Parameterized DML Statements
   - 4.2 Non-Parameterized DML Statements

5. **Additional Information**

**Requirements**

This tutorial assumes that you have already connected to the server (see Connecting to PostgreSQL), created the necessary objects on the server (see Creating Database Objects), and inserted data into tables (see Inserting Data Into Tables). To delete data at runtime, add the PgAccess unit to the uses clause for Delphi or include the PgAccess.hpp header file for C++ Builder.

**General Information**

You can delete data from a table using the Data Manipulation Language (DML), which is part of SQL. The user must have the appropriate privileges to execute DML statements on the server. There are two ways to manipulate data in a table: you can build DML statements manually and run them with a component like TPgQuery, or you can use the dataset functionality (the Delete method) of the TPgQuery and TPgTable components. Both ways are covered in this tutorial. This tutorial shows you how to delete a record from the dept table.

**Using the DataSet Functionality**

The Delete method of the TPgQuery and TPgTable components allows you to delete data without having to manually construct a DML statement — it is generated by PgDAC components internally. The code below demonstrates the use of this method:

**Delphi**

```delphi
var
  PgQuery1: TPgQuery;
begin
  PgQuery1 := TPgQuery.Create(nil);
  try
    // PgConnection1 was set up earlier
    PgQuery1.Connection := PgConnection1;
    // adds a statement to retrieve data
    PgQuery1.SQL.Text := 'SELECT * FROM dept';
    // opens the dataset
    PgQuery1.Open;
    // deletes the active record
    PgQuery1.Delete;
  finally
    PgQuery1.Free;
  end;
```

© 2021 Devart
Getting Started

C++Builder

TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
    // PgConnection1 was set up earlier
    PgQuery1->Connection = PgConnection1;
    // adds a statement to retrieve data
    PgQuery1->SQL->Text = "SELECT * FROM dept";
    // opens the dataset
    PgQuery1->Open();
    // deletes the active record
    PgQuery1->Delete();
}
__finally {
    PgQuery1->Free();
}

Building DML Statements Manually

DML statements can be constructed with or without parameters. The code below demonstrates both ways.

Parameterized DML Statements

Delphi

var
    PgQuery1: TPgQuery;
begin
    PgQuery1 := TPgQuery.Create(nil);
    try
        // PgConnection1 was set up earlier
        PgQuery1.Connection := PgConnection1;
        // adds a statement to delete a record
        PgQuery1.SQL.Add('DELETE FROM dept WHERE deptno = :deptno;');
        // searches parameters by their names and assigns new values
        PgQuery1.ParamByName('deptno').AsInteger := 10;
        // executes the statement
        PgQuery1.Execute;
    finally
        PgQuery1.Free;
    end;
end;

C++Builder

TIBCQuery* PgQuery1 = new TIBCQuery(NULL);
try {
    // PgConnection1 was set up earlier
    PgQuery1->Connection = PgConnection1;
    // adds a statement to delete a record
    PgQuery1->SQL->Add("DELETE FROM dept WHERE deptno = :deptno;" );
    // searches parameters by their names and assigns new values
PgQuery1->ParamByName("deptno")->AsInteger = 10;
// executes the statement
PgQuery1->Execute();
}
__finally {
    PgQuery1->Free();
}
}

Non-Parameterized DML Statements

Delphi

var
    PgQuery1: TPgQuery;
begin
    PgQuery1 := TPgQuery.Create(nil);
    try
        // PgConnection1 was set up earlier
        PgQuery1.Connection := PgConnection1;
        // adds a statement to delete a record
        PgQuery1.SQL.Add('DELETE FROM dept WHERE deptno = 10;');
        // executes the statement
        PgQuery1.Execute;
    finally
        PgQuery1.Free;
    end;
end;

C++Builder

TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
    // PgConnection1 was set up earlier
    PgQuery1->Connection = PgConnection1;
    // adds a statement to delete a record
    PgQuery1->SQL->Add("DELETE FROM dept WHERE deptno = 10; ");
    // executes the statement
    PgQuery1->Execute();
}__finally {
    PgQuery1->Free();
}

Additional Information

It is also possible to use stored procedures to delete data, in which case all data manipulation logic is defined on the server. See Using Stored Procedures for more information.
3.4 Creating Database Objects

This tutorial describes how to create database objects in PostgreSQL using the **TPgSQL** and **TPgScript** components.

1. Requirements
2. General Information
3. Creating Tables
   - 3.1 Design-Time
   - 3.2 Runtime
4. Creating Stored Procedures
   - 4.1 Design-Time
   - 4.2 Runtime
5. Additional Information

Requirements

This tutorial assumes that you have already connected to the server (see Connecting to PostgreSQL). To create database objects at runtime, add the **PgAccess** and **PgScript** units to the `uses` clause for Delphi or include the `PgAccess.hpp` and `PgScript.hpp` header files for C++ Builder.

General Information

Database objects are created using Data Definition Language (DDL), which is part of the SQL language. The user must have the appropriate privileges to execute DDL statements on the server. There are two ways to create database objects: build DDL statements manually and execute them with a component like **TPgSQL**, or use a GUI tool like pgAdmin. This tutorial uses the data access components to create tables and stored procedures.

Creating Tables

To create tables, the **TPgSQL** component is used in this tutorial.

Design-Time

- Find the **TPgSQL** component in the PgDAC category on the Tool Palette.
- Double-click the component. A new object will appear on the form. If this is the first **TPgSQL** object in this project, it will be named `PgSQL1`. Note that the **Connection** property is automatically set to an existing connection.
- Double-click the `PgSQL1` object.
- Enter the following statements:
CREATE TABLE dept (
    deptno serial not null,
    dname varchar(14),
    loc varchar(13),
    primary key (deptno)
);

CREATE TABLE emp (
    empno serial not null,
    ename varchar(10),
    job varchar(9),
    mgr integer,
    hiredate timestamp,
    sal int,
    comm int,
    deptno int references dept,
    primary key (empno)
);
Getting Started

C++Builder

TPgSQL * PgSQL1 = new TPgSQL(NULL);

try {
  // PgConnection1 was set up earlier
  PgSQL1->Connection = PgConnection1;
  // adds statements to create tables
  PgSQL1->SQL->Add("CREATE TABLE dept (");
  PgSQL1->SQL->Add(" deptno serial not null, ");
  PgSQL1->SQL->Add(" dname varchar(14), ");
  PgSQL1->SQL->Add(" loc varchar(13), ");
  PgSQL1->SQL->Add(" primary key (deptno) ");
  PgSQL1->SQL->Add(" )");
  PgSQL1->SQL->Add(" CREATE TABLE emp (");
  PgSQL1->SQL->Add(" empno serial not null, ");
  PgSQL1->SQL->Add(" ename varchar(10), ");
  PgSQL1->SQL->Add(" job varchar(9), ");
  PgSQL1->SQL->Add(" mgr int, ");
  PgSQL1->SQL->Add(" hiredate timestamp, ");
  PgSQL1->SQL->Add(" sal int, ");
  PgSQL1->SQL->Add(" comm int, ");
  PgSQL1->SQL->Add(" deptno int references dept (deptno) ");
  PgSQL1->SQL->Add(" )");
  // executes the statements
  PgSQL1->Execute();
}
__finally {
  PgSQL1->Free();
}

Creating Stored Procedures

To create stored procedures, the TPgScript component is used in this tutorial.

Design-Time

- Find the TPgScript component in the PgDAC category on the Tool Palette.
- Double-click the component. A new object will appear on the form. If this is the first TPgScript object in this project, it will be named PgScript1. Note that the Connection property is already set to an existing connection.
- Double-click the PgScript1 object.
- Enter the following statement:

CREATE FUNCTION TenMostHighPaidEmployees() RETURNS SETOF Emp AS $$
  SELECT * FROM emp ORDER BY emp.sal DESC LIMIT 10;$$;
$$ LANGUAGE sql;
CREATE FUNCTION GetEmpNumberInDept(
    IN  pdeptno int,
    OUT pempnumb int)
RETURNS int AS $$
BEGIN
    pempnumb := (SELECT count(*) FROM emp WHERE deptno = :pdeptno);
END;
$$ LANGUAGE plpgsql;

• Click the **Execute** button to create two stored procedures.

### Runtime

The same stored procedures created at runtime:

#### Delphi

```pascal
var
    PgScript1: TPgScript;
begin
    PgScript1 := TPgScript.Create(nil);
    try
        // PgConnection1 was set up earlier
        PgScript1.Connection := con;
        //adds statements to create procedures
        PgScript1.SQL.Add('CREATE FUNCTION TenMostHighPaidEmployees()');
        PgScript1.SQL.Add('RETURNS SetOf Emp AS $$');
        PgScript1.SQL.Add(' SELECT * FROM emp ORDER BY emp.sal DESC LIMIT 10');
        PgScript1.SQL.Add('$$ LANGUAGE sql;');
        PgScript1.SQL.Add('');
        PgScript1.SQL.Add('CREATE FUNCTION GetEmpNumberInDept(');
        PgScript1.SQL.Add(' IN  pdeptno int,');
        PgScript1.SQL.Add(' OUT pempnumb int');
        PgScript1.SQL.Add(' RETURNS int AS $$');
        PgScript1.SQL.Add('BEGIN');
        PgScript1.SQL.Add(' pempnumb := (SELECT count(*) FROM emp WHERE deptno = :pdeptno)
        PgScript1.SQL.Add('END;');
        PgScript1.SQL.Add('$$ LANGUAGE plpgsql;');
        // executes the statements
        PgScript1.Execute;
    finally
        PgScript1.Free;
    end;
end;
```

#### C++Builder

```cpp
TPgScript* PgScript1 = new TPgScript(NULL);
try {
    // PgConnection1 was set up earlier
    PgScript1->Connection = con;
    //adds statements to create procedures
    PgScript1->SQL->Add("CREATE FUNCTION TenMostHighPaidEmployees()");
Additional Information

There are many ways to create database objects on the server. Any tool or component that is capable of running an SQL query can be used to manage database objects. For example, `TPgSQL` can be used to insert statements one by one, while `TPgScript` is intended to execute multiple DDL/DML statements as a single SQL script. For more information on DDL statements, refer to the PostgreSQL documentation.

© 1997-2021 Devart. All Rights Reserved.

3.5 Inserting Data Into Tables

This tutorial describes how to insert data into tables using the `TPgQuery` and `TPgTable` components.

1. Requirements
2. General Information
3. Design-Time
4. Runtime
   - 4.1 Using the DataSet Functionality
   - 4.2 Building DML Statements Manually
     - 4.2.1 Parameterized DML Statements
     - 4.2.2 Non-Parameterized DML Statements
5. Additional Information

Requirements
This tutorial assumes that you already know how to connect to the server (see Connecting to PostgreSQL) and that the necessary objects have already been created on the server (see Creating Database Objects). To insert data at runtime, add the PgAccess unit to the uses clause for Delphi or include the PgAccess.hpp header file for C++ Builder.

General Information

You can insert data into a table using the Data Manipulation Language (DML), which is part of SQL. The user must have the appropriate privileges to execute DML statements on the server. There are two ways to manipulate data in a table: you can build DML statements manually and run them with a component like TPgQuery, or you can use the dataset functionality (the Insert, Append, and Post methods) of the TPgQuery and TPgTable components. This tutorial shows you how to insert data into the dept and emp tables.

The **dept** table definition:

<table>
<thead>
<tr>
<th>deptno</th>
<th>dname</th>
<th>loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
</tr>
<tr>
<td>30</td>
<td>SALE</td>
<td>CHICAGO</td>
</tr>
<tr>
<td>40</td>
<td>OPERATIONS</td>
<td>BOSTON</td>
</tr>
</tbody>
</table>

The **emp** table definition:

<table>
<thead>
<tr>
<th>empno</th>
<th>ename</th>
<th>job</th>
<th>mgr</th>
<th>hiredate</th>
<th>sal</th>
<th>comm</th>
<th>deptno</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>CLERK</td>
<td>7902</td>
<td>17-12-1980</td>
<td>800</td>
<td>NULL</td>
<td>20</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>SALESMAN</td>
<td>7698</td>
<td>20-02-1981</td>
<td>1600</td>
<td>300</td>
<td>30</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>SALESMAN</td>
<td>7698</td>
<td>22-02-1981</td>
<td>1250</td>
<td>500</td>
<td>30</td>
</tr>
<tr>
<td>ID</td>
<td>Name</td>
<td>Position</td>
<td>Manager</td>
<td>Hire Date</td>
<td>Salary</td>
<td>Bonus</td>
<td>Commission</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>------------</td>
<td>---------</td>
<td>---------------</td>
<td>--------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>7566</td>
<td>Jones</td>
<td>Manager</td>
<td>7839</td>
<td>02-04-1981</td>
<td>2975</td>
<td>NULL</td>
<td>20</td>
</tr>
<tr>
<td>7654</td>
<td>Martin</td>
<td>Salesman</td>
<td>7698</td>
<td>28-09-1981</td>
<td>1250</td>
<td>1400</td>
<td>30</td>
</tr>
<tr>
<td>7698</td>
<td>Blake</td>
<td>Manager</td>
<td>7839</td>
<td>01-05-1981</td>
<td>2850</td>
<td>NULL</td>
<td>30</td>
</tr>
<tr>
<td>7782</td>
<td>Clark</td>
<td>Manager</td>
<td>7839</td>
<td>09-06-1981</td>
<td>2450</td>
<td>NULL</td>
<td>10</td>
</tr>
<tr>
<td>7788</td>
<td>Scott</td>
<td>Analyst</td>
<td>7566</td>
<td>13-07-1987</td>
<td>3000</td>
<td>NULL</td>
<td>20</td>
</tr>
<tr>
<td>7839</td>
<td>King</td>
<td>President</td>
<td>NULL</td>
<td>17-11-1981</td>
<td>5000</td>
<td>NULL</td>
<td>10</td>
</tr>
<tr>
<td>7844</td>
<td>Turner</td>
<td>Salesman</td>
<td>7698</td>
<td>08-09-1981</td>
<td>1500</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>7876</td>
<td>Adams</td>
<td>Clerk</td>
<td>7788</td>
<td>13-07-1987</td>
<td>1100</td>
<td>NULL</td>
<td>20</td>
</tr>
<tr>
<td>7900</td>
<td>James</td>
<td>Clerk</td>
<td>7698</td>
<td>03-12-1981</td>
<td>950</td>
<td>NULL</td>
<td>30</td>
</tr>
<tr>
<td>7902</td>
<td>Ford</td>
<td>Analyst</td>
<td>7566</td>
<td>03-12-1981</td>
<td>3000</td>
<td>NULL</td>
<td>20</td>
</tr>
<tr>
<td>7934</td>
<td>Miller</td>
<td>Clerk</td>
<td>7782</td>
<td>23-01-1982</td>
<td>1300</td>
<td>NULL</td>
<td>10</td>
</tr>
</tbody>
</table>

**Inserting at Design-Time**

- Find the TPgQuery component in the PgDAC category on the Tool Palette.
- Double-click the component. A new object will appear on the form. If this is the first time that you create a TPgQuery object in this application, the object will be named PgQuery1. Note that the PgQuery1.Connection property is automatically set to an existing connection.
- Double-click the PgQuery1 object.
- Enter the following statement:

```sql
INSERT INTO dept VALUES (10,'ACCOUNTING','NEW YORK');
```
• Click the **Execute** button to add a new record to the **dept** table.

### Inserting at Runtime

#### Using the DataSet Functionality

The **Insert**, **Append**, and **Post** methods of the **TPgQuery** and **TPgTable** components allow you to insert data without having to manually construct a DML statement — it is generated by **PgDAC** components internally. **Insert** adds a new empty record in the current cursor position, while **Append** adds a new empty record at the end of the dataset. The code below demonstrates the use of these methods:

**Delphi**

```delphi
var
  PgQuery1: TPgQuery;
begin
  PgQuery1 := TPgQuery.Create(nil);
  try
    // PgConnection1 was set up earlier
    PgQuery1.Connection := PgConnection1;
    // adds a statement to retrieve data
    PgQuery1.SQL.Text := 'SELECT * FROM dept';
    // opens the dataset
    PgQuery1.Open;
    // adds a new empty record at the end of the dataset
    PgQuery1.Append;
    // searches fields by their names and assigns new values
    PgQuery1.FieldByName('deptno').AsInteger := 10;
    PgQuery1.FieldByName('dname').AsString := 'ACCOUNTING';
    PgQuery1.FieldByName('loc').AsString := 'NEW YORK';
    // writes the modified record
    PgQuery1.Post;
  finally
    PgQuery1.Free;
  end;
end;
```

**C++Builder**

```c++
TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
  // PgConnection1 was set up earlier
  PgQuery1->Connection = PgConnection1;
  // adds a statement to retrieve data
  PgQuery1->SQL->Text = "SELECT * FROM dept";
  // opens the dataset
  PgQuery1->Open();
  // adds a new empty record at the end of the dataset
  PgQuery1->Append();
  // searches fields by their names and assigns new values
  PgQuery1->FieldByName("deptno")->AsInteger = 10;
  PGQuery1->FieldByName("dname")->AsString = "ACCOUNTING";
  PGQuery1->FieldByName("loc")->AsString = "NEW YORK";
  // writes the modified record
  PgQuery1->Post();
} finally {
  PgQuery1->Free();
}
```

© 2021 Devart
Building DML Statements Manually

DML statements can be constructed with or without parameters. The code below demonstrates both ways.

Parameterized DML Statements

Delphi

```delphi
var
  PgQuery1: TPgQuery;
begin
  PgQuery1 := TPgQuery.Create(nil);
  try
    // PgConnection1 was set up earlier
    PgQuery1.Connection := PgConnection1;
    // adds a parameterized statement to insert data
    PgQuery1.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (:deptno, :dname, :loc));
    // searches parameters by their names and assigns new values
    PgQuery1.ParamByName('deptno').AsInteger := 10;
    PgQuery1.ParamByName('dname').AsString := 'ACCOUNTING';
    PgQuery1.ParamByName('loc').AsString := 'NEW YORK';
    // executes the statement
    PgQuery1.Execute;
  finally
    PgQuery1.Free;
  end;
end;
```

C++Builder

```c++
TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
  // PgConnection1 was set up earlier
  PgQuery1->Connection = PgConnection1;
  // adds a parameterized statement to insert data
  PgQuery1->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (:deptno, :dname, :loc);
  // searches parameters by their names and assigns new values
  PgQuery1->ParamByName("deptno")->AsInteger = 10;
  PgQuery1->ParamByName("dname")->AsString = "ACCOUNTING";
  PgQuery1->ParamByName("loc")->AsString = "NEW YORK";
  // executes the statement
  PgQuery1->Execute();
}
finally {
  PgQuery1->Free();
}
```
Non-Parameterized DML Statements

Delphi

```delphi
var PgQuery1: TPgQuery;
begin
  PgQuery1 := TPgQuery.Create(nil);
  try
    // PgConnection1 was set up earlier
    PgQuery1.Connection := PgConnection1;
    // adds a statement to insert a record
    PgQuery1.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (10, 'ACCOUNTING', 'NEW YORK');');
    // executes the statement
    PgQuery1.Execute;
  finally
    PgQuery1.Free;
  end;
end;
```

C++Builder

```cpp
TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
  // PgConnection1 was set up earlier
  PgQuery1->Connection = PgConnection1;
  // adds the statement to insert a record
  PgQuery1->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (10, 'ACCOUNTING', 'NEW YORK');");
  // executes the statement
  PgQuery1->Execute();
}
finally {
  PgQuery1->Free();
}
```

Additional Information

There are many ways to insert data into tables. Any tool or component that is capable of running an SQL query can be used to manage data. For example, **TPgSQL** can be used to insert records one by one, while **TPgScript** is designed to execute multiple DDL/DML statements as a single SQL script. **TPgLoader** is the fastest way to insert data into PostgreSQL tables.

It is also possible to use stored procedures to insert data, in which case all data manipulation logic is defined on the server. See [Using Stored Procedures](#) for more information.
3.6 Retrieving Data From Tables

This tutorial describes how to retrieve data from a table using the **TPgQuery** and **TPgTable** components.

1. Requirements
2. General Information
3. TPgQuery
4. TPgTable
5. Additional Information

Requirements

This tutorial assumes that you have already connected to the server (see [Connecting to PostgreSQL](#)), created the necessary database objects (see [Creating Database Objects](#)), and inserted data into tables (see [Inserting Data Into Tables](#)). To retrieve data at runtime, add the **PgAccess** unit to the `uses` clause for Delphi or include the `PgAccess.hpp` header file for C++ Builder.

General Information

PgDAC provides the **TPgQuery** and **TPgTable** components for retrieving data from a table. This tutorial shows you how to retrieve data from the `dept` table.

TPgQuery

The following code demonstrates how to retrieve data from the `dept` table using **TPgQuery**:

```delphi
var
  PgQuery1: TPgQuery;
begin
  PgQuery1 := TPgQuery.Create(nil);
  try
    // PgConnection1 was set up earlier
    PgQuery1.Connection := PgConnection1;
    // adds a statement to retrieve data
    PgQuery1.SQL.Text := 'SELECT * FROM dept';
    // opens the dataset
    PgQuery1.Open;
    // shows the number of records in the dataset
    ShowMessage(IntToStr(PgQuery1.RecordCount));
    finally
      PgQuery1.Free;
    end;
```
end;

**C++Builder**

```cpp
TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
    // PgConnection1 was set up earlier
    PgQuery1->Connection = PgConnection1;
    // adds a statement to retrieve data
    PgQuery1->SQL->Text = "SELECT * FROM dept";
    // opens the dataset
    PgQuery1->Open();
    // shows the number of records in the dataset
    ShowMessage(IntToStr(PgQuery1->RecordCount));
}
__finally {
    PgQuery1->Free();
}
```

**TPgTable**

The following code demonstrates how to retrieve data from the `dept` table using `TPgTable`:

**Delphi**

```delphi
var
    PgTable1: TPgTable;
begin
    PgTable1 := TPgTable.Create(nil);
    try
        // PgConnection1 was set up earlier
        PgTable1.Connection := PgConnection1;
        // indicates the name of the table
        PgTable1.TableName := 'dept';
        // opens the dataset
        PgTable1.Open;
        // shows the number of records in the dataset
        ShowMessage(IntToStr(PgTable1.RecordCount));
    finally
        PgTable1.Free;
    end;
end;
```

**C++Builder**

```cpp
TPgTable* PgTable1 = new TPgTable(NULL);
try {
    // PgConnection1 was set up earlier
    PgTable1->Connection = PgConnection1;
    // indicates the name of the table
    PgTable1->TableName = "dept";
    // opens the dataset
    PgTable1->Open();
    // shows the number of records in the dataset
    ShowMessage(IntToStr(PgTable1->RecordCount));
}
Additional Information

It is also possible to use stored procedures to delete data, in which case all data manipulation logic is defined on the server. See Using Stored Procedures for more information.

3.7 Modifying Data in Tables

This tutorial describes how to modify data in tables using the TPgQuery and TPgTable components.

1. Requirements
2. General Information
3. Using the DataSet Functionality
4. Building DML Statements Manually
   o 4.1 Parameterized DML Statements
   o 4.2 Non-Parameterized DML Statements
5. Additional Information

Requirements

This tutorial assumes that you have already connected to the server (see Connecting to PostgreSQL), created the necessary objects on the server (see Creating Database Objects), and inserted data into tables (see Inserting Data Into Tables). To modify data at runtime, add the PgAccess unit to the uses clause for Delphi or include the PgAccess.hpp header file for C++ Builder.

General Information

You can modify data in a table using the Data Manipulation Language (DML), which is part of SQL. DML statements can be executed on the server by a user with respective privileges. There are two ways to manipulate data in a table: you can build DML statements manually and run them with a component like TPgQuery, or you can use the dataset functionality (the Edit and Post methods) of the TPgQuery and TPgTable components. This tutorial shows you how to modify data in the dept table:

10 ACCOUNTING NEW YORK
Using the DataSet Functionality

The Edit and Post methods of the TPgQuery and TPgTable components allows you to modify data without having to manually construct a DML statement — it is generated by PgDAC components internally. The code below demonstrates the use of these methods:

**Delphi**

```delphi
var
  PgQuery1: TPgQuery;
begin
  PgQuery1 := TPgQuery.Create(nil);
  try
    // PgConnection1 was set up earlier
    PgQuery1.Connection := PgConnection1;
    // adds a statement to retrieve data
    PgQuery1.SQL.Text := 'SELECT * FROM dept';
    // opens the dataset
    PgQuery1.Open;
    // positions the cursor on the deptno=10 record
    PgQuery1.FindKey([10]);
    // enables editing of data in the dataset
    PgQuery1.Edit;
    // searches fields by their names and assigns new values
    PgQuery1.FieldByName('dname').AsString := 'RESEARCH';
    PgQuery1.FieldByName('loc').AsString := 'LOS ANGELES';
    // writes the modified record
    PgQuery1.Post;
  finally
    PgQuery1.Free;
  end;
end;
```

**C++Builder**

```cpp
TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
  // PgConnection1 was set up earlier
  PgQuery1->Connection = PgConnection1;
  // adds a statement to retrieve data
  PgQuery1->SQL->Text = "SELECT * FROM dept";
  // opens the dataset
  PgQuery1->Open();
  // positions the cursor on the deptno=10 record
  PgQuery1->FindKey(ARRAYOFCONST((10)));
  // enables editing of data in the dataset
  PgQuery1->Edit();
  // searches fields by their names and assigns new values
  PgQuery1->FieldByName("dname")->AsString = "RESEARCH";
```
Parameterized DML Statements

**Delphi**

```delphi
var
  PgQuery1: TPQuery;
begin
  PgQuery1 := TPgQuery.Create(nil);
  try
    // PgConnection1 was set up earlier
    PgQuery1.Connection := PgConnection1;
    // adds a statement to update a record
    PgQuery1.SQL.Add('UPDATE dept SET dname = :dname, loc = :loc WHERE deptno = :deptno');
    // searches parameters by their names and assigns new values
    PgQuery1.ParamByName('deptno').AsInteger := 10;
    PgQuery1.ParamByName('dname').AsString := 'RESEARCH';
    PgQuery1.ParamByName('loc').AsString := 'LOS ANGELES';
    // executes the statement
    PgQuery1.Execute;
  finally
    PgQuery1.Free;
  end;
end;
```

**C++Builder**

```cpp
TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
  // PgConnection1 was set up earlier
  PgQuery1->Connection = PgConnection1;
  // adds a statement to update a record
  PgQuery1->SQL->Add("UPDATE dept SET dname = :dname, loc = :loc WHERE deptno = :deptno");
  // searches parameters by their names and assigns new values
  PgQuery1->ParamByName("deptno")->AsInteger = 10;
  PgQuery1->ParamByName("dname")->AsString = "RESEARCH";
  PgQuery1->ParamByName("loc")->AsString = "LOS ANGELES";
  // executes the statement
  PgQuery1->Execute();
}
finally {
  PgQuery1->Free();
}
```
Non-Parameterized DML Statements

**Delphi**

```delphi
var
  PgQuery1: TPgQuery;
begin
  PgQuery1 := TPgQuery.Create(nil);
  try
    // PgConnection1 was set up earlier
    PgQuery1.Connection := PgConnection1;
    // adds the statement to update a record
    PgQuery1.SQL.Add('UPDATE dept SET dname = ''RESEARCH'', loc = ''LOS ANGELES'' WHERE deptno = 10;');
    // executes the statement
    PgQuery1.Execute;
  finally
    PgQuery1.Free;
  end;
end;
```

**C++Builder**

```cpp
TPgQuery* PgQuery1 = new TPgQuery(NULL);
try {
  // PgConnection1 was set up earlier
  PgQuery1->Connection = PgConnection1;
  // adds a statement to update a record
  PgQuery1->SQL->Add("UPDATE dept SET dname = 'RESEARCH', loc = 'LOS ANGELES' WHERE deptno = 10;" );
  // executes the statement
  PgQuery1->Execute();
}__finally {
  PgQuery1->Free();
}
```

**Additional Information**

It is also possible to use stored procedures to delete data, in which case all data manipulation logic is defined on the server. See [Using Stored Procedures](#) for more information.

---

3.8 **Using Stored Procedures**

This tutorial describes how to work with stored procedures using the [TPgStoredProc](#) component to insert data into tables.

1. **Requirements**
2. **General information**
3. **Input Parameters**
4. **Output Parameters**
5. **Input/Output Parameters**
6. **Using PostgreSQL Stored Functions**
7. **Returning Result Sets**

**Requirements**

This tutorial assumes that you have already connected to the server (see [Connecting to PostgreSQL](#), created the necessary objects on the server (see [Creating Database Objects](#)), and inserted data into tables (see [Inserting Data Into Tables](#)). To insert data at runtime, add the **PgAccess** unit to the `uses` clause for Delphi or include the **PgAccess.hpp** header file for C++ Builder.

**General Information**

A **stored procedure** is a group of one or more SQL statements grouped as a logical unit and stored in the database. Stored procedures are intended to perform a specific task or a set of related tasks. They combine the ease and flexibility of the SQL language with the procedural functionality of a structured programming language. Complicated business rules and programming logic that may require execution of multiple SQL statements should be kept in stored procedures, which can be called by the client applications.

A **stored function** is similar to a stored procedure, but there are some differences: a function must return a value, whereas in a stored procedure it is optional; a function can have only input parameters, whereas a procedures can have input or output parameters; a function can be called from a procedure, whereas a procedure cannot be called from a function.

**Input parameters**

**Input parameters** are used to pass values from the calling program to the stored procedure or function. If the procedure changes the input value, the change has effect only within the procedure, and the input variable will preserve its original value when control is returned to the calling program. The following procedure inserts a new row into the table **dept**:

```sql
CREATE PROCEDURE InsertDept(  p_deptno int,  p_dname varchar(14),  p_loc varchar(13))AS $$  INSERT INTO dept(deptno, dname, loc) VALUES(:p_deptno, :p_dname, :p_loc);$$ LANGUAGE sql;
```
The procedure accepts three input arguments that correspond to the fields of the table, and can be executed as follows:

CALL InsertDept(10,'ACCOUNTING','NEW YORK');

The code below demonstrates the use of the TPgStoredProc component to execute the InsertDept stored procedure:

**Delphi**

```delphi
var
  PgStoredProc1: TPgStoredProc;
begin
  PgStoredProc1 := TPgStoredProc.Create(nil);
  try
    // PgConnection1 was set up earlier
    PgStoredProc1.Connection := PgConnection1;
    // indicates the name of the stored procedure to call
    PgStoredProc1.StoredProcName := 'InsertDept';
    // constructs a statement based on the Params and StoredProcName
    // properties, and assigns it to the SQL property
    PgStoredProc1.PrepareSQL;
    // searches parameters by their names and assigns new values
    PgStoredProc1.ParamByName('p_deptno').AsInteger := 10;
    PgStoredProc1.ParamByName('p_dname').AsString := 'ACCOUNTING';
    PgStoredProc1.ParamByName('p_loc').AsString := 'NEW YORK';
    // executes the stored procedure
    PgStoredProc1.Execute;
  finally
    PgStoredProc1.Free;
  end;
end;
```

**C++Builder**

```cpp
TPgStoredProc* PgStoredProc1 = new TPgStoredProc(NULL);
try {
  // PgConnection1 was set up earlier
  PgStoredProc1->Connection = PgConnection1;
  // indicates the name of the stored procedure to call
  PgStoredProc1->StoredProcName = "InsertDept";
  // constructs a statement based on the Params and StoredProcName
  // properties, and assigns it to the SQL property
  PgStoredProc1->PrepareSQL();
  // searches parameters by their names and assigns new values
  PgStoredProc1->ParamByName("p_deptno")->AsInteger = 10;
  PgStoredProc1->ParamByName("p_dname")->AsString = "ACCOUNTING";
  PgStoredProc1->ParamByName("p_loc")->AsString = "NEW YORK";
  // executes the stored procedure
  PgStoredProc1->Execute();
}
__finally {
  PgStoredProc1->Free();
}
```
Output Parameters

Output parameters are used to return values from the procedure or function to the calling application. The initial value of the parameter in the procedure is NULL, and the value becomes visible to the calling program only when the procedure returns it. PostgreSQL supports output parameters only in stored functions. The following stored function returns the count of records in the `dept` table:

```sql
CREATE FUNCTION CountDept(cnt int) RETURNS int AS $$
BEGIN
    RETURN (SELECT count(*) FROM dept);
END;
$$ LANGUAGE plpgsql;
```

The code below demonstrates the use of the `TIBCStoredProc` component to execute the `CountDept` stored procedure:

**Delphi**

```delphi
var
    IBCStoredProc1: TIBCStoredProc;
begin
    IBCStoredProc1 := TIBCStoredProc.Create(nil);
    try
        // IBCConnection1 was set up earlier
        IBCStoredProc1.Connection := IBCConnection1;
        // indicates the name of the stored procedure to call
        IBCStoredProc1.StoredProcName := 'CountDept';
        // constructs a statement based on the Params and StoredProcName properties, and assigns it to the SQL property
        IBCStoredProc1.PrepareSQL;
        // executes the stored procedure
        IBCStoredProc1.Execute;
        // shows the value of the output parameter
        ShowMessage(IntToStr(sp.ParamByName('cnt').AsInteger));
    finally
        IBCStoredProc1.Free;
    end;
end;
```

**C++Builder**

```cpp
TIBCSavedProc* IBCStoredProc1 = new TIBCSavedProc(NULL);
try {
    // IBCConnection1 was set up earlier
    IBCStoredProc1->Connection = IBCConnection1;
    // indicates the name of the stored procedure to call
    IBCSavedProc1->StoredProcName = "CountDept";
    // constructs a statement based on the Params and StoredProcName properties, and assigns it to the SQL property
```
Input/output parameters

Input/output parameters (INOUT) act as the IN or OUT parameter, or both. Programs can pass a value to the stored procedure or function, which changes it and returns the updated value. The input value must be set before executing the stored procedure. INOUT parameters act like an initialized variables.

The following stored procedure returns the salary with a 5% percent bonus:

```sql
CREATE PROCEDURE GiveBonus(INOUT sal real)
AS $$
BEGIN
    sal = sal * 1.05;
END;
$$ LANGUAGE plpgsql;
```

The code below demonstrates the use of the `TPgStoredProc` component to execute the `GiveBonusFunc` stored function:

**Delphi**

```delphi
var
    PgStoredProc1: TPgStoredProc;
begin
    PgStoredProc1 := TPgStoredProc.Create(nil);
    try
        // PgConnection1 was set up earlier
        PgStoredProc1.Connection := PgConnection1;
        // indicates the name of the stored procedure to call
        PgStoredProc1.StoredProcName := 'GiveBonus';
        // constructs a statement based on the Params and StoredProcName
        // properties, and assigns it to the SQL property
        PgStoredProc1.PrepareSQL;
        // searches a parameter by its name and assigns a new value
        PgStoredProc1.ParamByName('sal').AsFloat := 500.5;
        // executes the stored procedure
        PgStoredProc1.Execute;
        // shows the resulting value
        ShowMessage(PgStoredProc1.ParamByName('sal').AsString);
    finally
        PgStoredProc1.Free;
    end;
end;
```
C++Builder

TPgStoredProc* PgStoredProc1 = new TPgStoredProc(NULL);
try {
    // PgConnection1 was set up earlier
    PgStoredProc1->Connection = PgConnection1;
    // indicates the name of the stored procedure to call
    PgStoredProc1->StoredProcName = "GiveBonus";
    // constructs a statement based on the Params and StoredProcName properties, and assigns it to the SQL property
    PgStoredProc1->PrepareSQL();
    // searches a parameter by its name and assigns a new value
    PgStoredProc1->ParamByName("sal")->AsFloat = 500.5;
    // executes the stored procedure
    PgStoredProc1->Execute();
    // shows the resulting value
    ShowMessage(PgStoredProc1->ParamByName("sal")->AsString);
}
__finally {
    PgStoredProc1->Free();
}

Using PostgreSQL Stored Functions

The tasks above can also be accomplished using stored functions in PostgreSQL. For example, the following stored function returns the salary with a 5% percent bonus:

CREATE FUNCTION GiveBonusFunc(sal real)
RETURNS real AS $$
BEGIN
    RETURN sal * 1.05;
END;
$$ LANGUAGE plpgsql;

This function can be executed as follows:

SELECT GiveBonusFunc(500.5);

The code below demonstrates the use of the TPgStoredProc component to execute the GiveBonusFunc stored function:

Delphi

var
    PgStoredProc1: TPgStoredProc;
begin
    PgStoredProc1 := TPgStoredProc.Create(nil);
    try
        // PgConnection1 was set up earlier
        PgStoredProc1.Connection := PgConnection1;
        // indicates the name of the stored procedure to call
        PgStoredProc1.StoredProcName := 'GiveBonusFunc';
        // constructs a statement based on the Params and StoredProcName properties, and assigns it to the SQL property
        PgStoredProc1->PrepareSQL();
        // searches a parameter by its name and assigns a new value
        PgStoredProc1->ParamByName("sal")->AsFloat = 500.5;
        // executes the stored procedure
        PgStoredProc1->Execute();
        // shows the resulting value
        ShowMessage(PgStoredProc1->ParamByName("sal")->AsString);
    __finally {
        PgStoredProc1->Free();
    }

PgStoredProc1.PrepareSQL;
    // searches a parameter by its name and assigns a new value
    PgStoredProc1.ParamByName('sal').AsFloat := 500.5;
    // executes the stored procedure
    PgStoredProc1.Execute;
    // shows the resulting value
    ShowMessage(PgStoredProc1.ParamByName('result').AsString);
finally
    IBCStoredProc1.Free;
end;
end;

C++Builder

TPgStoredProc* PgStoredProc1 = new TPgStoredProc(NULL);
try {
    // PgConnection1 was set up earlier
    PgStoredProc1->Connection = PgConnection1;
    // indicates the name of the stored procedure to call
    PgStoredProc1->StoredProcName = "GiveBonusFunc";
    // constructs a statement based on the Params and StoredProcName
    // properties, and assigns it to the SQL property
    PgStoredProc1->PrepareSQL();
    // searches a parameter by its name and assigns a new value
    PgStoredProc1->ParamByName("sal")->AsFloat = 500.5;
    // executes the stored procedure
    PgStoredProc1->Execute();
    // shows the resulting value
    ShowMessage(PgStoredProc1->ParamByName("result")->AsString);
}
__finally {
    PgStoredProc1->Free();
}

Note: To retrieve the result returned by the stored function using TPgStoredProc, use the automatically created 'result' parameter.

Returning Result Sets

Besides scalar variables, a stored function can return a result set generated by the SELECT statement. See Using Stored Functions with Result Sets for more information.

3.9 Using Stored Functions with Result Sets

This tutorial describes how to retrieve and modify result sets obtained from stored functions using the TPgStoredProc component
Requirements

This tutorial assumes that you have already connected to the server (see Connecting to PostgreSQL), created the necessary objects on the server (see Creating Database Objects), and inserted data into tables (see Inserting Data Into Tables). To insert data at runtime, add the PgAccess unit to the uses clause for Delphi or include the PgAccess.hpp header file for C++ Builder.

General Information

Besides scalar variables, a stored function can return a result set generated by the SELECT statement. You can insert or modify data in a result set using the dataset functionality of the TPgStoredProc component.

This tutorial shows you how to retrieve and modify data in the dept table using the TPgStoredProc component. The following stored procedure will be used to retrieve data:

```sql
CREATE FUNCTION TenMostHighPaidEmployees() RETURNS SETOF Emp AS $$
    SELECT * FROM dept;
$$ LANGUAGE sql;
```

Using the DataSet Functionality

The Insert, Append, Edit, and Post methods of the TPgStoredProc component can be used to insert and modify data without having to manually construct a DML statement — it is generated by PgDAC components internally. The code below demonstrates the use of these methods:

**Delphi**

```delphi
var
    PgStoredProc11: TPgStoredProc;
begin
    PgStoredProc1 := TPgStoredProc.Create(nil);
    try
        // PgConnection1 was set up earlier
        PgStoredProc1.Connection := PgConnection1;
        // indicates the name of the stored procedure to call
        PgStoredProc1.StoredProcName := 'SelectDept';
        // constructs a statement based on the Params and StoredProcName
        // properties, and assigns it to the SQL property
        PgStoredProc1.PrepareSQL;
        // opens the dataset
        PgStoredProc1.Open;
        // adds a new empty record at the end of the dataset
        PgStoredProc1.Append;
    except
        on E: Exception do
            // handle exceptions
    end;
```
// searches fields by their names and assigns new values
PgStoredProc1.FieldByName('deptno').AsInteger := 50;
PgStoredProc1.FieldByName('dname').AsString := 'SALES';
PgStoredProc1.FieldByName('loc').AsString := 'NEW YORK';
// writes the modified record
PgStoredProc1.Post;
// adds a new empty record in the current cursor position
PgStoredProc1.Insert;
PgStoredProc1.FieldByName('deptno').AsInteger := 60;
PgStoredProc1.FieldByName('dname').AsString := 'ACCOUNTING';
PgStoredProc1.FieldByName('loc').AsString := 'LOS ANGELES';
PgStoredProc1.Post;
// positions the cursor on the deptno=10 record
PgStoredProc1.FindKey([10]);
// enables editing of data in the dataset
PgStoredProc1(Edit);
PgStoredProc1.FieldByName('dname').AsString := 'RESEARCH';
PgStoredProc1.FieldByName('loc').AsString := 'LOS ANGELES';
PgStoredProc1.Post;
finally
  PgStoredProc1.Free;
end;
end;

C++Builder

TPgStoredProc* PgStoredProc1 = new TPgStoredProc(NULL);
try {
  // PgConnection1 was set up earlier
  PgStoredProc1->Connection = PgConnection1;
  // indicates the name of the stored procedure to call
  PgStoredProc1->StoredProcName = "SelectDept";
  // constructs a statement based on the Params and StoredProcName
  // properties, and assigns it to the SQL property
  PgStoredProc1->PrepareSQL();
  // opens the dataset
  PgStoredProc1->Open();
  // adds a new empty record at the end of the dataset
  PgStoredProc1->Append();
  // searches fields by their names and assigns new values
  PgStoredProc1->FieldByName("deptno")->AsInteger = 50;
  PgStoredProc1->FieldByName("dname")->AsString = "SALES";
  PgStoredProc1->FieldByName("loc")->AsString = "NEW YORK";
  // writes the modified record
  PgStoredProc1->Post();
  // adds a new empty record in the current cursor position
  PgStoredProc1->Insert();
  PgStoredProc1->FieldByName("deptno")->AsInteger = 60;
  PgStoredProc1->FieldByName("dname")->AsString = "ACCOUNTING";
  PgStoredProc1->FieldByName("loc")->AsString = "LOS ANGELES";
  PgStoredProc1->Post();
  // positions the cursor on the deptno=10 record
  PgStoredProc1->FindKey(ARRAYOFCONST((10)));
  // enables editing of data in the dataset
  PgStoredProc1->Edit();
  PgStoredProc1->FieldByName("dname")->AsString = "RESEARCH";
  PgStoredProc1->FieldByName("loc")->AsString = "LOS ANGELES";
3.10 Demo Projects

PgDAC includes a number of demo projects that show off the main PgDAC functionality and development patterns.

The PgDAC demo projects consist of one large project called PgDacDemo with demos for all main PgDAC components, use cases, and data access technologies, and a number of smaller projects on how to use PgDAC in different IDEs and how to integrate PgDAC with third-party components.

Most demo projects are built for Delphi and Embarcadero RAD Studio. There are only two PgDAC demos for C++Builder. However, the C++Builder distribution includes source code for all the other demo projects as well.

Where are the PgDAC demo projects located?

In most cases all the PgDAC demo projects are located in "%PgDac%\Demos".

In Delphi 2007 for Win32 under Windows Vista all the PgDAC demo projects are located in "My Documents\Devart\PgDac for Delphi 2007\Demos", for example "C:\Documents and Settings\All Users\Documents\Devart\PgDac for Delphi 2007\Demos\".

The structure of the demo project directory depends on the IDE version you are using.

For most new IDEs the structure will be as follows.

Demos

|-- PgDacDemo [The main PgDAC demo project]
  |-- TechnologySpecific
    |-- SecureBridge [A component and a demo for integration with the SecureBridge library]
    |-- ThirdParty
      |-- [A collection of demo projects on integration with third-party components]
    |-- Miscellaneous
PgDacDemo is the main demo project that shows off all the PgDAC functionality. The other directories contain a number of supplementary demo projects that describe special use cases. A list of all the samples in the PgDAC demo project and a description for the supplementary projects is provided in the following section.

Note: This documentation describes ALL the PgDAC demo projects. The actual demo projects you will have installed on your computer depends on your PgDAC version, PgDAC edition, and the IDE version you are using. The integration demos may require installation of third-party components to compile and work properly.

Instructions for using the PgDAC demo projects

To explore a PgDAC demo project,
1. Launch your IDE.
2. In your IDE, choose File | Open Project from the menu bar.
3. Find the directory you installed PgDAC to and open the Demos folder.
4. Browse through the demo project folders located here and open the project file of the demo you would like to use.
5. Compile and launch the demo. If it exists, consult the ReadMe.txt file for more details.

The included sample applications are fully functional. To use the demos, you have to first set up a connection to PostgreSQL. You can do so by clicking on the "Connect" button.

Many demos may also use some database objects. If so, they will have two object manipulation buttons, "Create" and "Drop". If your demo requires additional objects, click "Create" to create the necessary database objects. When you are done with a demo, click "Drop" to remove all the objects used for the demo from your database.

Note: The PgDAC demo directory includes two sample SQL scripts for creating and dropping all the test schema objects used in the PgDAC demos. You can modify and execute this script manually, if you would like. This will not change the behavior of the demos.

You can find a complete walkthrough for the main PgDAC demo project in the Getting Started topic. The other PgDAC demo projects include a ReadMe.txt file with individual building and launching instructions.

Demo project descriptions
**PgDacDemo**

*PgDacDemo* is one large project which includes three collections of demos.

**Working with components**

A collection of samples that show how to work with the basic PgDAC components.

**General demos**

A collection of samples that show off the PgDAC technology and demonstrate some ways to work with data.

**PostgreSQL-specific demos**

A collection of samples that demonstrate how to incorporate PostgreSQL features in database applications.

*PgDacDemo* can be opened from `%PgDac%\Demos\PgDacDemo\PgDacDemo.dpr` (.bdsproj). The following table describes all demos contained in this project.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alerter</td>
<td>Uses <code>TPgAlerter</code> to send notifications between connections.</td>
</tr>
<tr>
<td>ConnectDialog</td>
<td>Demonstrates how to customize the <a href="#">PgDAC connect dialog</a> . Changes the standard PgDAC connect dialog to two custom connect dialogs. The first customized sample dialog is inherited from the TForm class, and the second one is inherited from the default PgDAC connect dialog class.</td>
</tr>
<tr>
<td>CRDBGrid</td>
<td>Demonstrates how to work with the TCRDBGrid component. Shows off the main TCRDBGrid features, like filtering, searching, stretching, using compound headers, and more.</td>
</tr>
<tr>
<td>Dump</td>
<td>Demonstrates how to backup data from tables with the <code>TPgDump</code> component. Shows how to use scripts created during back up to restore table data. This demo lets you back up a table either by specifying the table name or by writing a SELECT query.</td>
</tr>
<tr>
<td>Loader</td>
<td>Uses the <code>TPgLoader</code> component to quickly load data into a server table. This demo also compares the two <code>TPgLoader</code> data loading handlers: <code>GetColumnData</code> and <code>PutData</code>.</td>
</tr>
<tr>
<td>Query</td>
<td>Demonstrates working with <code>TPgQuery</code>, which is one of the most useful PgDAC components. Includes many <code>TPgQuery</code> usage scenarios. Demonstrates how to execute queries in both standard and NonBlocking mode and how to edit data and export it to XML files.</td>
</tr>
</tbody>
</table>
**Note:** This is a very good introductory demo. We recommend starting here when first becoming familiar with PgDAC.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sql</td>
<td>Uses TPgSQL to execute SQL statements. Demonstrates how to work in a separate thread, in standard mode, in NonBlocking mode, and how to break long-duration query execution.</td>
</tr>
<tr>
<td>StoredProc</td>
<td>Uses TPgStoredProc to access an editable recordset from a PostgreSQL stored procedure in the client application.</td>
</tr>
<tr>
<td>Table</td>
<td>Demonstrates how to use TPgTable to work with data from a single table on the server without writing any SQL queries manually. Performs server-side data sorting and filtering and retrieves results for browsing and editing.</td>
</tr>
<tr>
<td>UpdateSQL</td>
<td>Demonstrates using the TPgUpdateSQL component to customize update commands. Lets you optionally use T:Devart.PgDac.TPgCommand and TPgQuery objects for carrying out insert, delete, query, and update commands.</td>
</tr>
<tr>
<td>VirtualTable</td>
<td>Demonstrates working with the TVirtualTable component. This sample shows how to fill virtual dataset with data from other datasets, filter data by a given criteria, locate specified records, perform file operations, and change data and table structure.</td>
</tr>
</tbody>
</table>

### General Demos

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>Demonstrates how to perform the most important tasks of working with data in CachedUpdates mode, including highlighting uncommitted changes, managing transactions, and committing changes in a batch.</td>
</tr>
<tr>
<td>FilterAndIndex</td>
<td>Demonstrates PgDAC's local storage functionality. This sample shows how to perform local filtering, sorting and locating by multiple fields, including by calculated and lookup fields.</td>
</tr>
<tr>
<td>MasterDetail</td>
<td>Uses PgDAC functionality to work with master/detail relationships. This sample shows how to use local master/detail functionality. Demonstrates different kinds of master/detail linking, including linking by SQL, by simple fields, and by calculated fields.</td>
</tr>
<tr>
<td>Lock</td>
<td>Demonstrates the recommended approach for managing transactions with the TPgConnection component. The TPgConnection interface provides a wrapper for PostgreSQL server commands like START TRANSACTION, COMMIT, ROLLBACK.</td>
</tr>
</tbody>
</table>

### PostgreSQL-specific Demos
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pictures</td>
<td>Uses PgDAC functionality to work with graphics. The sample demonstrates how to retrieve binary data from PostgreSQL server database and display it on visual components. Sample also shows how to load and save pictures to files and to the database.</td>
</tr>
<tr>
<td>Text</td>
<td>Uses PgDAC functionality to work with text. The sample demonstrates how to retrieve text data from SQL Server database and display it on visual components. Sample also shows how to load and save text to files and to the database.</td>
</tr>
</tbody>
</table>

**Supplementary Demo Projects**

PgDAC also includes a number of additional demo projects that describe some special use cases, show how to use PgDAC in different IDEs and give examples of how to integrate it with third-party components. These supplementary PgDAC demo projects are sorted into subfolders in the %PgDac\%\Demos\ directory.

<table>
<thead>
<tr>
<th>Location</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThirdParty</td>
<td>FastReport</td>
<td>Demonstrates how PgDAC can be used with FastReport components. This project consists of two parts. The first part is several packages that integrate PgDAC components into the FastReport editor. The second part is a demo application that lets you design and preview reports with PgDAC technology in the FastReport editor.</td>
</tr>
<tr>
<td>Technology Specific</td>
<td>SecureBridge</td>
<td>The demo project demonstrates how to integrate the SecureBridge components with PgDAC to ensure secure connection to PostgreSQL server through an SSH tunnel and SSL. This demo consists of three parts. The first part is a package that contains the TPgSSHIOHandler and TPgSSLIOHandler component. These components provide integration with the SecureBridge library. The second part is two sample projects that demonstrate how to connect to PostgreSQL server through an SSH server and through SSL, connect to the SSH server with SecureBridge by password or by public key, generate reliable random numbers, enable local port forwarding. For more information see the Readme.html file in the demo directory.</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>Dll</td>
<td>Demonstrates creating and loading DLLs for PgDAC-based projects. This demo project consists of two parts - an Pg_Dll project that creates a DLL of a form that sends a query to the server and displays its results, and an Pg_Exe project that</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>can be executed</td>
<td>can be executed to display a form for loading and running this DLL. Allows you to build a dll for one PgDAC-based project and load and test it from a separate application.</td>
<td></td>
</tr>
<tr>
<td>FailOver</td>
<td>Demonstrates the recommended approach to working with unstable networks. This sample lets you perform transactions and updates in several different modes, simulate a sudden session termination, and view what happens to your data state when connections to the server are unexpectedly lost. Shows off CachedUpdates, LocalMasterDetail, FetchAll, Pooling, and different Failover modes.</td>
<td></td>
</tr>
</tbody>
</table>

| PgDacDemo        | PgDacDemo[Win32 version of the main PgDAC demo project - see above]                                                                                                                                              |

3.11 Deployment

PgDAC applications can be built and deployed with or without run-time libraries. Using run-time libraries is managed with the "Build with runtime packages" check box in the Project Options dialog box.

Deploying Windows applications built without run-time packages

You do not need to deploy any files with PgDAC-based applications built without run-time packages, provided you are using a registered version of PgDAC.

You can check if your application does not require run-time packages by making sure the "Build with runtime packages" check box is not selected in the Project Options dialog box.

Trial Limitation Warning

If you are evaluating deploying Windows applications with PgDAC Trial Edition, you will need to deploy the following DAC BPL files:

- `dacXX.bpl`  always
- `pgdacXX.bpl` always

and their dependencies (required IDE BPL files) with your application, even if it is built without run-time packages:

- `rtlXX.bpl` always
Deploying Windows applications built with run-time packages

You can set your application to be built with run-time packages by selecting the "Build with runtime packages" check box in the Project Options dialog box before compiling your application.

In this case, you will also need to deploy the following BPL files with your Win32 application:

<table>
<thead>
<tr>
<th>BPL File</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbrtlXX.bpl</td>
<td>always</td>
</tr>
<tr>
<td>vclodbXXX.bpl</td>
<td>always</td>
</tr>
<tr>
<td>dacXX.bpl</td>
<td>always</td>
</tr>
<tr>
<td>pgdacXX.bpl</td>
<td>always</td>
</tr>
<tr>
<td>dacvclXX.bpl</td>
<td>if your application uses the PgDacVcl unit</td>
</tr>
<tr>
<td>pgdacvclXX.bpl</td>
<td>if your application uses the PgDacVcl unit</td>
</tr>
<tr>
<td>crcontrolsXX.bpl</td>
<td>if your application uses the CRDBGrid component</td>
</tr>
</tbody>
</table>

4 Using PgDAC

This section describes basics of using PostgreSQL Data Access Components

- Updating Data with PgDAC Dataset Components
- Master/Detail Relationships
- Automatic Key Field Value Generation
- Data Type Mapping
- Data Encryption
- Working in an Unstable Network
- Secure Connections
-Disconnected Mode
- Increasing Performance
- Macros
- DataSet Manager
- TPgLoader Component
- Large Objects
- REFCURSOR Data Type
4.1 Updating Data with PgDAC Dataset Components

PgDAC dataset components which descend from TCustomDADataset provide different ways for reflecting local changes on the server.

The first approach is to use automatic generation of update SQL statements. When using this approach you should specify Key Fields (the KeyFields property) to avoid requesting KeyFields from the server. When SELECT statement uses multiple tables, you can use the T:Devart.PgDac.TCustomPgDataSet.UpdatingTable property to specify which table will be updated. If UpdatingTable is blank, the first table of the FROM clause will be used. In the most cases PgDAC needs an additional information about updating objects. So PgDAC executes additional queries to the server. This helps to generate correct updating SQL statements but may result in performance decrease. To disable these additional queries, set the ExtendedFieldsInfo option to False.

Another approach is to set update SQL statements using SQLInsert, SQLUpdate, and SQLDelete properties. Use them to specify SQL statements that will be used for corresponding data modifications. It is useful when generating data modification statements is not possible (for example, when working with data returned by a stored procedure) or you need to execute some specific statements. You may also assign the TPgUpdateSQL component to the UpdateObject property. TPgUpdateSQL component holds all updating SQL statements in one place. You can generate all these SQL statements using PgDAC design time editors. For more careful customization of data update operations you can use InsertObject, ModifyObject and DeleteObject properties of the TPgUpdateSQL component.

See Also
4.2 Master/Detail Relationships

Master/detail (MD) relationship between two tables is a very widespread one. So it is very important to provide an easy way for database application developer to work with it. Let's examine how PgDAC implements this feature.

Suppose we have classic MD relationship between "Department" and "Employee" tables.

"Department" table has field Dept_No. Dept_No is a primary key.

"Employee" table has a primary key EmpNo and foreign key Dept_No that binds "Employee" to "Department".

It is necessary to display and edit these tables.

PgDAC provides two ways to bind tables. First code example shows how to bind two TCustomPgDataSet components (TPgQuery, TPgTable, or TPgStoredProc) into MD relationship via parameters.

```pascal
procedure TForm1.Form1Create(Sender: TObject);
var  Master, Detail: TPgQuery;  MasterSource: TDataSource;
begin  // create ...
  Detail.MasterSource := MasterSource;  // open master dataset and only then detail dataset  Master.Open;  Detail.Open;
end;
```

Pay attention to one thing: parameter name in detail dataset SQL must be equal to the field name or the alias in the master dataset that is used as foreign key for detail table. After
opening detail dataset always holds records with Dept_No field value equal to the one in the current master dataset record.

There is an additional feature: when inserting new records to detail dataset it automatically fills foreign key fields with values taken from master dataset.

Now suppose that detail table "Department" foreign key field is named DepLink but not Dept_No. In such case detail dataset described in above code example will not autofill DepLink field with current "Department".Dept_No value on insert. This issue is solved in second code example.

```pascal
procedure TForm1.Form1Create(Sender: TObject);
var
  Master, Detail: TPgQuery;
  MasterSource: TDataSource;
begin
  // create master dataset
  Master := TPaquery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  // create detail dataset
  Detail := TPgQuery.Create(Self);
  Detail.SQL.Text := 'SELECT * FROM Employee';
  // setup MD
  Detail.MasterFields := 'DepLink'; // foreign key in Employee
  MasterSource := TDataSource.Create(Self);
  MasterSource.DataSet := Master;
  Detail.MasterSource := MasterSource;
  // open master dataset and only then detail dataset
  Master.Open;
  Detail.Open;
end;
```

In this code example MD relationship is set up using **MasterFields** and **DetailFields** properties. Also note that there are no WHERE clause in detail dataset SQL.

To defer refreshing of detail dataset while master dataset navigation you can use **DetailDelay** option.

Such MD relationship can be local and remote, depending on the **TCustomDADataSet.Options.LocalMasterDetail** option. If this option is set to True, dataset uses local filtering for establishing master-detail relationship and does not refer to the server. Otherwise detail dataset performs query each time when record is selected in master dataset. Using local MD relationship can reduce server calls number and save server resources. It can be useful for slow connection. **CachedUpdates** mode can be used for detail dataset only for local MD relationship. Using local MD relationship is not recommended when detail table contains too many rows, because in remote MD relationship only records that
correspond to the current record in master dataset are fetched. So, this can decrease
network traffic in some cases.

See Also
- TCustomDataObject.Options
- TMemDataSet.CachedUpdates

## 4.3 Automatic Key Field Value Generation

When editing a dataset it is often convenient to generate key field(s) values automatically
instead of filling them manually. In the most common way application developer generates
primary key value basing it on a previously created sequence. There are three ways of doing
it.

First, application independent way - developer uses SERIAL data type, or manually sets field
default value like the following:

```
ALTER TABLE Department ALTER COLUMN DepNo SET DEFAULT nextval('seq_deptno');
```

or creates AFTER INSERT trigger that fills the field value. But there he faces the problem
with getting inserted value back to dataset. This problem can be easily solved in PgDAC using
RETURNING clause. In order for dataset to return a field value specified in RETURNING
clause, set the TDADataSetOptions.ReturnParams property to True. For instance:

```
PgQuery.SQL.Text := 'SELECT DepNo, DepName, Location FROM Department';
PgQuery.SQLInsert.Text := 'INSERT INTO Department (DepNo, DepName, Location) ' +
                          'VALUES(DepNo, DepName, Location) ' +
                          'RETURNING DepNo';
PgQuery.Options.ReturnParams := True;
```

The second way is custom key field value generation. Developer can fill key field value in the
TCustomPgDataSet.BeforePost event handler. But in this case he should manually execute
query and retrieve the sequence value. So this way may be useful only if some special value
processing is needed.

The third way, using KeySequence, is the simplest. Developer only needs to specify two
properties and key field values are generated automatically. There is no need to create trigger
or perform custom BeforePost processing.

...
4.4 Data Type Mapping

Overview

Data Type Mapping is a flexible and easily customizable gear, which allows mapping between DB types and Delphi field types.

In this article there are several examples, which can be used when working with all supported DBs. In order to clearly display the universality of the Data Type Mapping gear, a separate DB will be used for each example.

Data Type Mapping Rules

In versions where Data Type Mapping was not supported, PgDAC automatically set correspondence between the DB data types and Delphi field types. In versions with Data Type Mapping support the correspondence between the DB data types and Delphi field types can be set manually.

Here is the example with the numeric type in the following table of a PostgreSQL database:

```sql
CREATE TABLE numeric_types
(
    id integer NOT NULL,
    value1 numeric(5,2),
    value2 numeric(10,4),
    value3 numeric(15,6),
    CONSTRAINT pk_numeric_types PRIMARY KEY (id)
)
```

And Data Type Mapping should be used so that:

- the numeric fields with Scale=0 in Delphi would be mapped to one of the field types: TSmallintField, TIntegerField or TlargeintField, depending on Precision
- to save precision, the numeric fields with Precision>=10 and Scale<= 4 would be mapped to TBCDField
• and the numeric fields with Scale >= 5 would be mapped to TFMTBCDField.

The above in the form of a table:

<table>
<thead>
<tr>
<th>PostgreSQL data type</th>
<th>Default Delphi field type</th>
<th>Destination Delphi field type</th>
</tr>
</thead>
<tbody>
<tr>
<td>numeric(4,0)</td>
<td>ftFloat</td>
<td>ftSmallint</td>
</tr>
<tr>
<td>numeric(10,0)</td>
<td>ftFloat</td>
<td>ftInteger</td>
</tr>
<tr>
<td>numeric(15,0)</td>
<td>ftFloat</td>
<td>ftLargeint</td>
</tr>
<tr>
<td>numeric(5,2)</td>
<td>ftFloat</td>
<td>ftFloat</td>
</tr>
<tr>
<td>numeric(10,4)</td>
<td>ftFloat</td>
<td>ftBCD</td>
</tr>
<tr>
<td>numeric(15,6)</td>
<td>ftFloat</td>
<td>ftFMTBCD</td>
</tr>
</tbody>
</table>

To specify that numeric fields with Precision <= 4 and Scale = 0 must be mapped to ftSmallint, such a rule should be set:

```pascal
var
  DBType: Word;
  MinPrecision: Integer;
  MaxPrecision: Integer;
  MinScale: Integer;
  MaxScale: Integer;
  FieldType: TFieldType;
begin
  DBType := pgNumeric;
  MinPrecision := 0;
  MaxPrecision := 4;
  MinScale := 0;
  MaxScale := 0;
  FieldType := ftSmallint;
  PgConnection.DataTypeMap.AddDBTypeRule(DBType, MinPrecision, MaxPrecision, MinScale, MaxScale, FieldType);
end;
```

This is an example of the detailed rule setting, and it is made for maximum visualization. Usually, rules are set much shorter, e.g. as follows:

```pascal
// clear existing rules
PgConnection.DataTypeMap.Clear;
// rule for numeric(4,0)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, 4, 0, 0, ftSmallint);
// rule for numeric(10,0)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 5, 10, 0, 0, ftInteger);
// rule for numeric(15,0)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 11, rlAny, 0, 0, ftLargeint);
// rule for numeric(5,2)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, 9, 1, rlAny, ftFloat);
// rule for numeric(10,4)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 10, rlAny, 1, 0, ftBCD);
// rule for numeric(15,6)
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 10, rlAny, 5, rlAny, ftFMTBCD);
```
Rules order

When setting rules, there can occur a situation when two or more rules that contradict to each other are set for one type in the database. In this case, only one rule will be applied — the one, which was set first.

For example, there is a table in an PostgreSQL database:

```sql
CREATE TABLE person
(
    id                integer      NOT NULL,
    firstname         character(20)        ,
    lastname          character(30)        ,
    gender_code       character(1)         ,
    birth_dttm        date                 ,
    CONSTRAINT pk_person_types PRIMARY KEY (id)
)
```

TBCDField should be used for NUMBER(10,4), and TFMTBCDField - for NUMBER(15,6) instead of default fields:

<table>
<thead>
<tr>
<th>PostgreSQL data type</th>
<th>Default Delphi field type</th>
<th>Destination field type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER(5,2)</td>
<td>ftFloat</td>
<td>ftFloat</td>
</tr>
<tr>
<td>NUMBER(10,4)</td>
<td>ftFloat</td>
<td>ftBCD</td>
</tr>
<tr>
<td>NUMBER(15,6)</td>
<td>ftFloat</td>
<td>ftFMTBCD</td>
</tr>
</tbody>
</table>

If rules are set in the following way:

```pascal
PgConnection.DataTypeMap.Clear;
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, 9, rlAny, rlAny, 0, rlAny, 0, 4, ftBCD);
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, rlAny, 0, rlAny, ftFMTBCD);
```

it will lead to the following result:

<table>
<thead>
<tr>
<th>PostgreSQL data type</th>
<th>Delphi field type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER(5,2)</td>
<td>ftFloat</td>
</tr>
<tr>
<td>NUMBER(10,4)</td>
<td>ftBCD</td>
</tr>
<tr>
<td>NUMBER(15,6)</td>
<td>ftFMTBCD</td>
</tr>
</tbody>
</table>

But if rules are set in the following way:

```pascal
PgConnection.DataTypeMap.Clear;
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, rlAny, 0, rlAny, ftBCD);
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, rlAny, 0, rlAny, 0, 4, ftBCD);
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, 9, rlAny, rlAny, ftBCD);
```
it will lead to the following result:

<table>
<thead>
<tr>
<th>PostgreSQL data type</th>
<th>Delphi field type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER(5,2)</td>
<td>ftFMTBCD</td>
</tr>
<tr>
<td>NUMBER(10,4)</td>
<td>ftFMTBCD</td>
</tr>
<tr>
<td>NUMBER(15,6)</td>
<td>ftFMTBCD</td>
</tr>
</tbody>
</table>

This happens because the rule

```csharp
PgConnection.DataTypeMap.AddDBTypeRule(pgNumeric, 0, rlAny, 0, rlAny, ftFMTBCD);
```

will be applied for the NUMBER fields, whose Precision is from 0 to infinity, and Scale is from 0 to infinity too. This condition is met by all NUMBER fields with any Precision and Scale.

When using Data Type Mapping, first matching rule is searched for each type, and it is used for mapping. In the second example, the first set rule appears to be the first matching rule for all three types, and therefore the ftFMTBCD type will be used for all fields in Delphi.

If to go back to the first example, the first matching rule for the NUMBER(5,2) type is the first rule, for NUMBER(10,4) - the second rule, and for NUMBER(15,6) - the third rule. So in the first example, the expected result was obtained.

So it should be remembered that if rules for Data Type Mapping are set so that two or more rules that contradict to each other are set for one type in the database, the rules will be applied in the specified order.

**Defining rules for Connection and Dataset**

Data Type Mapping allows setting rules for the whole connection as well as for each DataSet in the application.

For example, such table is created in SQL Server:

```sql
CREATE TABLE person
(
    id integer NOT NULL,
    firstname character(20),
    lastname character(30),
    gender_code character(1),
    birth_dttm date
)
CONSTRAINT pk_person_types PRIMARY KEY (id)
```
It is exactly known that the birth_dttm field contains birth day, and this field should be ftDate in Delphi, and not ftDateTime. If such rule is set:

```
PgConnection.DataTypeMap.Clear;
PgConnection.DataTypeMap.AddDBTypeRule(pgDate, ftDate);
```

all DATETIME fields in Delphi will have the ftDate type, that is incorrect. The ftDate type was expected to be used for the DATETIME type only when working with the person table. In this case, Data Type Mapping should be set not for the whole connection, but for a particular DataSet:

```
PgQuery.DataTypeMap.Clear;
PgQuery.DataTypeMap.AddDBTypeRule(pgDate, ftDate);
```

Or the opposite case. For example, DATETIME is used in the application only for date storage, and only one table stores both date and time. In this case, the following rules setting will be correct:

```
PgConnection.DataTypeMap.Clear;
PgConnection.DataTypeMap.AddDBTypeRule(pgDate, ftDate);
PgQuery.DataTypeMap.Clear;
PgQuery.DataTypeMap.AddDBTypeRule(pgDate, ftDateTime);
```

In this case, in all DataSets for the DATETIME type fields with the ftDate type will be created, and for PgQuery - with the ftDateTime type.

The point is that the priority of the rules set for the DataSet is higher than the priority of the rules set for the whole connection. This allows both flexible and convenient setting of Data Type Mapping for the whole application. There is no need to set the same rules for each DataSet, all the general rules can be set once for the whole connection. And if a DataSet with an individual Data Type Mapping is necessary, individual rules can be set for it.

### Rules for a particular field

Sometimes there is a need to set a rule not for the whole connection, and not for the whole dataset, but only for a particular field.

e.g. there is such table in a MySQL database:

```
CREATE TABLE item
(
    id integer NOT NULL,
    name character(50) NOT NULL,
    guid character(38),
    CONSTRAINT pk_item PRIMARY KEY (id)
)
The `guid` field contains a unique identifier. For convenient work, this identifier is expected to be mapped to the TGuidField type in Delphi. But there is one problem, if to set the rule like this:

```delphi
PgQuery.DataTypeMap.Clear;
PgQuery.DataTypeMap.AddDBTypeRule(pgCharacter, ftGuid);
```
then both name and guid fields will have the ftGuid type in Delphi, that does not correspond to what was planned. In this case, the only way is to use Data Type Mapping for a particular field:

```delphi
PgQuery.DataTypeMap.Clear;
PgQuery.DataTypeMap.AddFieldNameRule('guid', ftGuid)
```

In addition, it is important to remember that setting rules for particular fields has the highest priority. If to set some rule for a particular field, all other rules in the Connection or DataSet will be ignored for this field.

### Ignoring conversion errors

Data Type Mapping allows mapping various types, and sometimes there can occur the problem with that the data stored in a DB cannot be converted to the correct data of the Delphi field type specified in rules of Data Type Mapping or vice-versa. In this case, an error will occur, which will inform that the data cannot be mapped to the specified type.

For example:

<table>
<thead>
<tr>
<th>Database value</th>
<th>Destination field type</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>'text value'</td>
<td>ftInteger</td>
<td>String cannot be converted to Integer</td>
</tr>
<tr>
<td>1000000</td>
<td>ftSmallint</td>
<td>Value is out of range</td>
</tr>
<tr>
<td>15.1</td>
<td>ftInteger</td>
<td>Cannot convert float to integer</td>
</tr>
</tbody>
</table>

But when setting rules for Data Type Mapping, there is a possibility to ignore data conversion errors:

```delphi
PgConnection.DataTypeMap.AddDBTypeRule(pgCharacter, ftInteger, True);
```

In this case, the correct conversion is impossible. But because of ignoring data conversion errors, Data Type Mapping tries to return values that can be set to the Delphi fields or DB fields depending on the direction of conversion.
### 4.5 Data Encryption

PgDAC has built-in algorithms for data encryption and decryption. To enable encryption, you should attach the `TCREncryptor` component to the dataset, and specify the encrypted fields. When inserting or updating data in the table, information will be encrypted on the client side in accordance with the specified method. Also when reading data from the server, the components decrypt the data in these fields "on the fly".

For encryption, you should specify the data encryption algorithm (the `EncryptionAlgorithm` property) and password (the `Password` property). On the basis of the specified password, the key is generated, which encrypts the data. There is also a possibility to set the key directly using the `SetKey` method.

When storing the encrypted data, in addition to the initial data, you can also store additional information: the GUID and the hash. (The method is specified in the `TCREncryptor.DataHeader` property).

If data is stored without additional information, it is impossible to determine whether the data is encrypted or not. In this case, only the encrypted data should be stored in the column, otherwise, there will be confusion because of the inability to distinguish the nature of the data. Also in this way, the similar source data will be equivalent in the encrypted form, that is not good from the point of view of the information protection. The advantage of this method is the size of the initial data equal to the size of the encrypted data.

<table>
<thead>
<tr>
<th>Database value</th>
<th>Destination field type</th>
<th>Result</th>
<th>Result description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'text value'</td>
<td>ftInteger</td>
<td>0</td>
<td>0 will be returned if the text cannot be converted to number</td>
</tr>
<tr>
<td>1000000</td>
<td>ftSmallint</td>
<td>32767</td>
<td>32767 is the max value that can be assigned to the Smallint data type</td>
</tr>
<tr>
<td>15,1</td>
<td>ftInteger</td>
<td>15</td>
<td>15,1 was truncated to an integer value</td>
</tr>
</tbody>
</table>

Therefore ignoring of conversion errors should be used only if the conversion results are expected.
To avoid these problems, it is recommended to store, along with the data, the appropriate GUID, which is necessary for specifying that the value in the record is encrypted and it must be decrypted when reading data. This allows you to avoid confusion and keep in the same column both the encrypted and decrypted data, which is particularly important when using an existing table. Also, when doing in this way, a random initializing vector is generated before the data encryption, which is used for encryption. This allows you to receive different results for the same initial data, which significantly increases security.

The most preferable way is to store the hash data along with the GUID and encrypted information to determine the validity of the data and verify its integrity. In this way, if there was an attempt to falsify the data at any stage of the transmission or data storage, when decrypting the data, there will be a corresponding error generated. For calculating the hash the SHA1 or MD5 algorithms can be used (the HashAlgorithm property).

The disadvantage of the latter two methods - additional memory is required for storage of the auxiliary information.

As the encryption algorithms work with a certain size of the buffer, and when storing the additional information it is necessary to use additional memory, TCREncryptor supports encryption of string or binary fields only (ftString, ftWideString, ftBytes, ftVarBytes, ftBlob, ftMemo, ftWideMemo). If encryption of string fields is used, firstly, the data is encrypted, and then the obtained binary data is converted into hexadecimal format. In this case, data storage requires two times more space (one byte = 2 characters in hexadecimal).

Therefore, to have the possibility to encrypt other data types (such as date, number, etc.), it is necessary to create a field of the binary or BLOB type in the table, and then convert it into the desired type on the client side with the help of data mapping.

It should be noted that the search and sorting by encrypted fields become impossible on the server side. Data search for these fields can be performed only on the client after decryption of data using the Locate and LocateEx methods. Sorting is performed by setting the TMemDataSet.IndexFieldName property.

**Example.**

Let's say there is an employee list of an enterprise stored in the table with the following data: full name, date of employment, salary, and photo. We want all these data to be stored in the encrypted form. Write a script for creating the table:

```sql
CREATE TABLE emp (  
  empno integer,  
  ename character(2000),  
  hiredate date,  
  salary numeric,  
  photo bytea,  
)  
```
As we can see, the fields for storage of the textual information, date, and floating-point number are created with the VARBINARY type. This is for the ability to store encrypted information, and in the case of the text field - to improve performance. Write the code to process this information on the client.

```pascal
PgQuery.SQL.Text := 'SELECT * FROM emp';
PgQuery.Encryption.Encryptor := PgEncryptor;
PgQuery.Encryption.Fields := 'ename, hiredate, sal, foto';
PgEncryptor.Password := '111111';
PgQuery.DataTypeMap.AddFieldNameRule ('sal', ftFloat);
PgQuery.Open;
```

### 4.6 Working in an Unstable Network

The following settings are recommended for working in an unstable network:

- `TCustomDAConnection.Options.LocalFailover = True`
- `TCustomDAConnection.Options.DisconnectedMode = True`
- `TDataSet.CachedUpdates = True`
- `TCustomDADataSet.FetchAll = True`
- `TCustomDADataSet.Options.LocalMasterDetail = True`
- `AutoCommit = True`

These settings minimize the number of requests to the server. Using `TCustomDAConnection.Options.DisconnectedMode` allows DataSet to work without an active connection. It minimizes server resource usage and reduces connection break probability. I.e. in this mode connection automatically closes if it is not required any more. But every explicit operation must be finished explicitly. That means each explicit connect must be followed by explicit disconnect. Read [Working with Disconnected Mode](#) topic for more information.

Setting the `FetchAll` property to True allows to fetch all data after cursor opening and to close connection. If you are using master/detail relationship, we recommend to set the `LocalMasterDetail` option to True.

It is not recommended to prepare queries explicitly. Use the `CachedUpdates` mode for DataSet data editing. Use the `TCustomDADataSet.Options.UpdateBatchSize` property to
reduce the number of requests to the server.

If a connection breaks, a fatal error occurs, and the `OnConnectionLost` event will be raised if the following conditions are fulfilled:

- There are no active transactions;
- There are no opened and not fetched datasets;
- There are no explicitly prepared datasets or SQLs.

If the user does not refuse suggested RetryMode parameter value (or does not use the `OnConnectionLost` event handler), PgDAC can implicitly perform the following operations:

```
Connect;
DataSet.ApplyUpdates;
DataSet.Open;
```

I.e. when the connection breaks, implicit reconnect is performed and the corresponding operation is reexecuted. We recommend to wrap other operations in transactions and fulfill their reexecuting yourself.

The using of `Pooling` in Disconnected Mode allows to speed up most of the operations because of connecting duration reducing.

**See Also**

- FailOver demo
- Working with Disconnected Mode
- `TCustomDAConnection.Options`
- `TCustomDAConnection.Pooling`

© 1997-2021

Devar. All Rights Reserved.

### 4.7 Secure Connections

This section describes the ways to secure connection between a client and a PostgreSQL server

- Connecting via SSL
- Connecting via SSH
- HTTP/HTTPS Network Tunneling

© 1997-2021

Devar. All Rights Reserved.
4.7.1 Connecting via SSL

Connecting to PostgreSQL Through SSL in Delphi

Security is very important when sending messages from the server to the client and vice versa. There are many data protection methods, including the use of SSL encryption to connect to a remote PostgreSQL server from a Delphi application. PostgreSQL supports data transfer via the TCP/IP protocol stack both using SSL encryption or without it.

Devart offers a solution called SecureBridge, which allows you to embed an SSL client into a Delphi or C++ Builder application to establish a secure connection to PostgreSQL server. This tutorial demonstrates how to create a sample Delphi application that connects to PostgreSQL using SSL as the encryption method.

Before connecting to PostgreSQL via SSL, create SSL certificates as explained in the PostgreSQL documentation and configure SSL parameters in postgresql.conf and pg.hba.conf files.

Sample Delphi app that connects to PostgreSQL using SSL

To create an SSL connection to PostgreSQL, PgDAC provides several values for the SSLOptions property. For this tutorial, the Mode property is set to smRequire, since it forces the application to only connect via SSL connection - if a connection attempt fails, an exception is raised.

Example of SSLOptions property set to smRequire:

```
PgConnection.SSLOptions.Mode := smRequire;
```

After installing PgDAC and SecureBridge software on your machine, install the TCRSSLOHandler component in RAD Studio to bind PgDAC with SecureBridge. The installation instructions are provided in the Readme.html, which is located by default in "My Documents\Devart\PgDAC for RAD Studio\Demos\TechnologySpecific\SecureBridge\DelphiXX".

2. Place the TCRSSLOHandler component, which allows PgDAC to connect to PostgreSQL server through SSL, onto the form. Also add the TPgConnection, TPgQuery, TDataSource, TDBGrid, and TButton components to the form - they are
required to create a sample application that connects to the PostgreSQL server via SSL, runs a selection operation against the database, and displays the obtained rows in a data grid.

3. Select the **TPgConnection** component and assign the **TCRSSLIOHandler** object to the **IOHandler** property in the Object Inspector.
4. Expand **SSLOptions** in the Object Inspector and specify the server certificate in the **CACert** property, the client certificate in the **Cert** property, and the private client key in the **Key** property.
5. Double-click **TPgConnection** and specify the server address, port, username, password, and, optionally, database name. Click **Connect** to test connection to the PostgreSQL server.
6. Select the **TDataSource** component and assign the **PgQuery1** object to the **DataSet** property.

7. Assign the **DataSource1** object to the **DataSource** property in the **TDBGrid** component.
8. Double-click the **TPgQuery** component and add a SQL query that will be run against the PostgreSQL database.

9. Select the **TButton** component and create the **OnClick** event. Add the code that will call the **Open** method in the **TPgQuery** component when you click the button.
10. Press F9 to compile and run the application. Click the button on the form to execute the query and display data in the grid.

2. SSL Connection to PostgreSQL in Delphi Using the OpenSSL Library

Another way to embed SSL client functionality into your Delphi app, which uses PgDAC components to access PostgreSQL, is by using the OpenSSL library that implements the SSL protocol and enables servers to securely communicate with their clients. The description of the SSL connection features without the SecureBridge IOHandler usage:

The following options must be set for SSL connection:
- **SSLCACert** - the pathname to the certificate authority file.
- **SSLCert** - the pathname to the certificate file.
- **SSLKey** - the pathname to the key file.
• SSLCipherList - a list of allowable ciphers to use for SSL encryption.

Note: The ssleay32.dll and libeay32.dll files are required to use the SSL protocol with the OpenSSL library.

© 1997-2021 Devart. All Rights Reserved.

4.7.2 Connecting via SSH

Connecting to PostgreSQL via SSH in Delphi

SSH is a protocol that allows users to securely log onto and interact with remote systems on the Internet by connecting a client program to an SSH server. SSH provides a mechanism for establishing a cryptographically secured connection between two endpoints, a client and a remote server, which authenticate each other and exchange messages. It employs different forms of symmetrical encryption, asymmetrical encryption, and hashing.

It is possible to use SSH to secure the network connection between a Delphi application and a PostgreSQL server. You execute shell commands in the same fashion as if you were physically operating the remote machine.

Devart offers a solution called SecureBridge that allows you to create a Delphi SSH client and a server. You can embed the SSH client into your application and install the SSH server on a remote machine where your PostgreSQL server resides. The SSH client connects to the SSH server, which sends all commands to the remote PostgreSQL server. This tutorial demonstrates how to create a sample Delphi application that connects to PostgreSQL using SSH as the encryption method.

SSH key-based authentication is done by public and private keys that a client uses to authenticate itself when logging into an SSH server. The server key is used is used by the client to authenticate the SSH server and is specified in the TScSSHClient.HostKeyName property. The client key is used by the SSH server to authenticate the client and is specified in the TScSSHClient.PrivateKeyName property. Note that the private key contains the public key. See SecureBridge tutorial on configuring the SSH server.

An SSH server is required to replicate the steps in this tutorial and encrypt the network
connection between the client application and the PostgreSQL server. You can build the SSH server demo project that is distributed with SecureBridge ('Documents\Devart\PgDAC for RAD Studio|Demos\TechnologySpecific\SecureBridge\Demo') and run the executable file.

After installing PgDAC and SecureBridge software on your system, install the TCRSSHIOHandler component in RAD Studio to bind PgDAC with SecureBridge. The installation instructions are provided in the Readme.html, which is located by default in "My Documents|Devart|PgDAC for RAD Studio XX|Demos\TechnologySpecific\SecureBridge".

Sample Delphi app that connects to PostgreSQL using SSH
2. Place the following components on the form: TCRSSHIOHandler, TPgConnection, TPgQuery, TScFileStorage, TScSSHClient, TDataSource, TDBGrid, and TButton.
   The sample application will connect to the PostgreSQL server via SSH, run a selection operation against the database, and display the obtained rows in the grid.
3. Select the TDBGrid and set the DataSource property to DataSource1.

4. In the TDataSource component, assign PgQuery1 to the DataSet property.
5. Select the TPgQuery and set the Connection property to PgConnection1. Double-click the component and enter a SQL statement to be executed against the PostgreSQL database.

6. Double-click the TButton to switch to the unit view. Add the code to call the Open method on the PgQuery1 object to activate the dataset when the button is clicked.
7. In the **TCRSSHIOHandler** component, assign **ScSSHClient1** to the **Client** property.

8. Select the **TScFileStorage** component and specify in the **Path** property the directory where keys are stored on your system. Double-click the component and **generate** a pair of keys for authenticating the server by the client.
9. Set the Authentication property to atPublicKey in the TScSSHClient component. In HostKeyName, specify the server public key. In PrivateKeyName, specify the client private key. The Hostname property holds the address of your server. Assign ScFileStorage1 to the KeyStorage property. Enter your username on the server in the User property. Specify the SSH port in the Port property.

10. Double-click the TPgConnection component. Specify your server address, port, database name (optionally), and username and password for the PostgreSQL user. Set the IOHandler property to CRSSHIOHandler1. Click Connect to check connection to the
11. Press F9 to compile and run the project. Click the button to run the query against the database and display the data in the form.

It is not obligatory to use SecureBridge TScSSHServer component as an SSH server - you can use any other server that implements the SSH protocol.
Network Tunneling

Usually when a client needs to connect to server it is assumed that direct connection can be established. Nowadays though, due to security reasons or network topology, it is often necessary to use a proxy or bypass a firewall. This article describes different ways to connect to PostgreSQL server with PgDAC.

- **Direct Connection**
- **Connection Through HTTP Tunnel**
  - Connection Through Proxy and HTTP Tunnel
- **Sample Delphi App That Connects to PostgreSQL Using HTTP/HTTPS Tunneling**
- **Additional Information**

Direct Connection

Direct connection to server means that server host is accessible from client without extra routing and forwarding. This is the simplest case. The only network setting you need is the host name and port number. This is also the fastest and most reliable way of communicating with server. Use it whenever possible.

The following code illustrates the simplicity:

```pascal
PgConnection := TPgConnection.Create(self);
PgConnection.Server := 'localhost';
PgConnection.Port := 5432;
PgConnection.Username := 'postgres';
PgConnection.Password := 'postgres';
PgConnection.Connect;
```

Connection Through HTTP Tunnel

Sometimes client machines are shielded by a firewall that does not allow you to connect to server directly at the specified port. If the firewall allows HTTP connections, you can use PgDAC together with HTTP tunneling software to connect to PostgreSQL server. PgDAC supports HTTP tunneling based on the PHP script.

An example of the web script tunneling usage can be the following: you have a remote website, and access to its database through the port of the database server is forbidden. Only access through HTTP port 80 is allowed, and you need to access the database from a remote computer, like when using usual direct connection.
You need to deploy the tunnel.php script, which is included into the provider package, on the web server. It allows access to the database server via HTTP tunneling. The script must be accessible through the HTTP protocol. You can verify if it is accessible with a web browser. The script can be found in the HTTP subfolder of the installed provider folder, e.g. %Program Files%\Devart\PgDac for Delphi XX\HTTP\tunnel.php. The only requirement to the server is PHP 5 support.

To connect to the database, you must set TPgConnection parameters for usual direct connection, which will be established from the web server side, the HttpOptions.Enabled property to True, and set the following parameters, specific for the HTTP tunneling:

<table>
<thead>
<tr>
<th>Property</th>
<th>Mandatory</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpOptions.Url</td>
<td>Yes</td>
<td>URL of the tunneling PHP script. For example, if the script is in the server root, the URL can be the following: <a href="http://localhost/tunnel.php">http://localhost/tunnel.php</a>.</td>
</tr>
<tr>
<td>HttpOptions.Username, HttpOptions.Password</td>
<td>No</td>
<td>Set these properties if the access to the website folder with the script is available only for registered users authenticated with user name and password.</td>
</tr>
</tbody>
</table>

Connecting Through Proxy and HTTP Tunnel

The HTTP tunneling server may be not directly accessible from the client machine. For example, client address is 10.0.0.2, server address is 192.168.0.10, and the PostgreSQL server listens on port 5433. The client and server reside in different networks, so the client can reach it only through proxy at address 10.0.0.1, which listens on port 808. In this case in addition to the TPgConnection.HttpOptions options you have to setup a HttpOptions.ProxyOptions object as follows:

```pascal
PgConnection := TPgConnection.Create(self);
PgConnection.Server := '192.168.0.10';
PgConnection.Port := 5433;
PgConnection.Username := 'postgres';
PgConnection.Password := 'postgres';
PgConnection.HttpOptions.Enabled := True;
PgConnection.HttpOptions.ProxyOptions.Hostname := '10.0.0.1';
PgConnection.HttpOptions.ProxyOptions.Port := 808;
PgConnection.HttpOptions.ProxyOptions.Username := 'ProxyUser';
PgConnection.HttpOptions.ProxyOptions.Password := 'ProxyPassword';
PgConnection.Connect;
```

Note that setting parameters of PgConnection.HttpOptions.ProxyOptions automatically
enables the use of of the proxy server.

Sample Delphi App That Connects to PostgreSQL Using HTTP/HTTPS Tunneling

- Open your browser and visit the URL of the tunnel.php script on your server to verify that the script has been properly installed.

- Run RAD Studio and select 'File -> New -> VCL Forms Application – Delphi'.
- Place the following components on the form: TPgConnection, TPgQuery, TDataSource, TDBGrid, TButton, and TCRSSLIOHandler. The last component is required when you connect via HTTPS. Note that the TCRSSLIOHandler is distributed with SecureBridge and is required to bind PgDAC with SecureBridge. The installation instructions for the component are provided in the Readme.html, which is located by default in "My Documents \Devart\PgDAC for RAD Studio\Demos\TechnologySpecific\SecureBridge\DelphiXX".

- Select TDBGrid and set the DataSource property to DataSource1.
In the TDataSource component, assign PgQuery1 to the DataSet property.

Select TPgQuery and set the Connection property to PgConnection1. Double-click the component and enter an SQL statement to be executed against the PostgreSQL database.
• Double-click **TButton** to switch to the unit view. Add code that invokes the **Open** method on the **PgQuery1** object to activate the dataset when the button is clicked.

• Select the **TPgConnection** component. If you use the secure tunnel, set the **IOHandler** property to **CRSSLIOHandler1**. Expand the **HttpOptions** and enter the URL of the **tunnel.php** script on your server. Set the **Enabled** property in **HttpOptions** to **True**.

• Double-click the **TPgConnection** component. Specify your server address, port, database name (optionally), username and password for the PostgreSQL user. Click **Connect** to test
connection to the PostgreSQL server.

- Press F9 to compile and run the project, and click the button to run the query against the database through HTTPS and display the data in the form.

Additional Information

There is one more way to tunnel network traffic. The Secure Shell forwarding, or SSH, can be used for forwarding data. However, main purpose of SSH is traffic encryption rather than avoiding firewalls or network configuration problems. The Connecting via SSH article describes how to use SSH protocol in PgDAC.

Keep in mind that traffic tunneling or encryption always increases CPU usage and network
load. It is recommended that you use direct connection whenever possible.

© 1997-2021 Devart. All Rights Reserved.

4.8 Disconnected Mode

In disconnected mode a connection opens only when it is required. After performing all server calls connection closes automatically until next server call is required. Datasets remain opened when connection closes. Disconnected Mode may be useful for saving server resources and operating in an unstable or expensive network. Drawback of using disconnected mode is that each connection establishing requires some time for authorization. If connection is often closed and opened it can slow down application work. We recommend to use pooling to solve this problem. For additional information see TCustomDAConnection.Pooling.

To enable disconnected mode set TCustomDAConnection.Options.DisconnectedMode to True.

In disconnected mode a connection is opened for executing requests to the server (if it was not opened already) and is closed automatically if it is not required any more. If the connection was explicitly opened (the Connect method was called or the Connected property was explicitly set to True), it does not close until the Disconnect method is called or the Connected property is set to False explicitly.

The following settings are recommended to use for working in disconnected mode:

TDataSet.CachedUpdates = True
TCustomDADataSet.FetchAll = True
TCustomDADataSet.Options.LocalMasterDetail = True

These settings minimize the number of requests to the server.

Disconnected mode features

If you perform a query with the FetchAll option set to True, connection closes when all data is fetched if it is not used by someone else. If the FetchAll option is set to false, connection does not close until all data blocks are fetched.

If explicit transaction was started, connection does not close until the transaction is committed or rolled back.

If the query was prepared explicitly, connection does not close until the query is unprepared or its SQL text is changed.
4.9 Batch Operations

Data amount processed by modern databases grows steadily. In this regard, there is an acute problem – database performance. Insert, Update and Delete operations have to be performed as fast as possible. Therefore Devart provides several solutions to speed up processing of huge amounts of data. So, for example, insertion of a large portion of data to a DB is supported in the TPgLoader. Unfortunately, TPgLoader allows to insert data only – it can't be used for updating and deleting data.

The new version of Devart Delphi Data Access Components introduces the new mechanism for large data processing — Batch Operations. The point is that just one parametrized Modify SQL query is executed. The plurality of changes is due to the fact that parameters of such a query will be not single values, but a full array of values. Such approach increases the speed of data operations dramatically. Moreover, in contrast to using TPgLoader, Batch operations can be used not only for insertion, but for modification and deletion as well.

Let's have a better look at capabilities of Batch operations with an example of the BATCH_TEST table containing attributes of the most popular data types.

Batch_Test table generating scripts

```sql
CREATE TABLE BATCH_TEST  
(  ID    INTEGER,  
  F_INTEGER INTEGER,  
  F_FLOAT DOUBLE PRECISION,  
  F_STRING VARCHAR(250),  
  F_DATE  DATE,  
  CONSTRAINT PK_BATCH_TEST PRIMARY KEY (ID) )
```
Batch operations execution

To insert records into the BATCH_TEST table, we use the following SQL query:

```
```

When a simple insertion operation is used, the query parameter values look as follows:

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>2.5</td>
<td>'String Value 1'</td>
<td>01.09.2015</td>
</tr>
</tbody>
</table>

After the query execution, one record will be inserted into the BATCH_TEST table.

When using Batch operations, the query and its parameters remain unchanged. However, parameter values will be enclosed in an array:

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>2.5</td>
<td>'String Value 1'</td>
<td>01.09.2015</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>3.15</td>
<td>'String Value 2'</td>
<td>01.01.2000</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>5.08</td>
<td>'String Value 3'</td>
<td>09.09.2010</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
<td>7.5343</td>
<td>'String Value 4'</td>
<td>10.10.2015</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>0.4555</td>
<td>'String Value 5'</td>
<td>01.09.2015</td>
</tr>
</tbody>
</table>

Now, 5 records are inserted into the table at a time on query execution.

How to implement a Batch operation in the code?

Batch INSERT operation sample

Let's try to insert 1000 rows to the BATCH_TEST table using a Batch Insert operation:

```delphi
var i: Integer;
begin
  // describe the SQL query
  PgQuery1.SQL.Text := 'INSERT INTO BATCH_TEST VALUES (:ID, :F_INTEGER, :F_FLOAT, :F_STRING, :F_DATE);
  // define the parameter types passed to the query:
  PgQuery1.Params[0].DataType := ftInteger;
  PgQuery1.Params[1].DataType := ftInteger;
  PgQuery1.Params[2].DataType := ftFloat;
  PgQuery1.Params[3].DataType := ftString;
  PgQuery1.Params[4].DataType := ftDateTime;
  // specify the array dimension:
  PgQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:
```
for i := 0 to PgQuery1.Params.ValueCount - 1 do begin
  PgQuery1.Params[0][i].AsInteger := i + 1;
  PgQuery1.Params[1][i].AsInteger := i + 2000 + 1;
  PgQuery1.Params[2][i].AsFloat := (i + 1) / 12;
  PgQuery1.Params[3][i].AsString := 'Values ' + IntToStr(i + 1);
  PgQuery1.Params[4][i].AsDateTime := Now;
end;

// insert 1000 rows into the BATCH_TEST table
PgQuery1.Execute(1000);
end;

This command will insert 1000 rows to the table with one SQL query using the prepared array of parameter values. The number of inserted rows is defined in the Iters parameter of the Execute(Iters: integer; Offset: integer = 0) method. In addition, you can pass another parameter – Offset (0 by default) – to the method. The Offset parameter points the array element, which the Batch operation starts from.

We can insert 1000 records into the BATCH_TEST table in 2 ways.

All 1000 rows at a time:

PgQuery1.Execute(1000);

2×500 rows:

// insert first 500 rows
PgQuery1.Execute(500, 0);
// insert next 500 rows
PgQuery1.Execute(500, 500);

500 rows, then 300, and finally 200:

// insert 500 rows
PgQuery1.Execute(500, 0);
// insert next 300 rows starting from 500
PgQuery1.Execute(300, 500);
// insert next 200 rows starting from 800
PgQuery1.Execute(200, 800);

Batch UPDATE operation sample

With Batch operations we can modify all 1000 rows of our BATCH_TEST table just this simple:

var
  i: Integer;
begin
  // describe the SQL query
  PgQuery1.SQL.Text := 'UPDATE BATCH_TEST SET F_INTEGER=:F_INTEGER, F_FLOAT=:F_FLOAT, ...
  // define parameter types passed to the query:
  PgQuery1.Params[0].DataType := ftInteger;
  PgQuery1.Params[1].DataType := ftFloat;
  PgQuery1.Params[2].DataType := ftString;
  PgQuery1.Params[3].DataType := ftDateTime;
Batch DELETE operation sample

Deleting 1000 rows from the BATCH_TEST table looks like the following operation:

```pascal
var
  i: Integer;
begin
  // describe the SQL query
  PgQuery1.SQL.Text := 'DELETE FROM BATCH_TEST WHERE ID=:ID';
  // define parameter types passed to the query:
  PgQuery1.Params[0].DataType := ftInteger;
  // specify the array dimension
  PgQuery1.Params.ValueCount := 1000;
  // populate the arrays with parameter values
  for i := 0 to 1000 - 1 do
    PgQuery1.Params[0][i].AsInteger := i + 1;
  end;
  // delete 1000 rows from the BATCH_TEST table
  PgQuery1.Execute(1000);
end;
```

Performance comparison

The example with BATCH_TEST table allows to analyze execution speed of normal operations with a database and Batch operations:

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>25 000 records</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard Operation (sec.)</td>
</tr>
<tr>
<td>Insert</td>
<td>346.7</td>
</tr>
<tr>
<td>Update</td>
<td>334.4</td>
</tr>
<tr>
<td>Delete</td>
<td>373.7</td>
</tr>
</tbody>
</table>

The less, the better.

It should be noted, that the retrieved results may differ when modifying the same table on different database servers. This is due to the fact that operations execution speed may differ
depending on the settings of a particular server, its current workload, throughput, network connection, etc.

**Thing you shouldn’t do when accessing parameters in Batch operations!**

When populating the array and inserting records, we accessed query parameters by index. It would be more obvious to access parameters by name:

```pascal
for i := 0 to 9999 do begin
  PgQuery1.Params.ParamByName('ID')[i].AsInteger := i + 1;
  PgQuery1.Params.ParamByName('F_INTEGER')[i].AsInteger := i + 2000 + 1;
  PgQuery1.Params.ParamByName('F_FLOAT')[i].AsFloat := (i + 1) / 12;
  PgQuery1.Params.ParamByName('F_STRING')[i].AsString := 'Values ' + IntToStr(i + 1);
  PgQuery1.Params.ParamByName('F_DATE')[i].AsDateTime := Now;
end;
```

However, the parameter array would be populated slower, since you would have to define the ordinal number of each parameter by its name in each loop iteration. If a loop is executed 10000 times – **performance loss can become quite significant.**

4.10 **Increasing Performance**

This topic considers basic stages of working with DataSet and ways to increase performance on each of these stages.

**Connect**

If your application performs Connect/Disconnect operations frequently, additional performance can be gained using pooling mode (TCustomDAConnection.Pooling = True). It reduces connection reopening time greatly (hundreds times). Such situation usually occurs in web applications.

**Execute**

If your application executes the same query several times, you can use the TCustomDADataSet.Prepare method or set the TDADataSetOptions.AutoPrepare property to increase performance. For example, it can be enabled for Detail dataset in Master/Detail relationship or for update objects in TCustomDAUpdateSQL. The performance gain achieved this way can be anywhere from several percent to several times, depending on the situation.

To execute SQL statements a TPgSQL component is more preferable than TPgQuery. It can
give several additional percents performance gain.

If the `TCustomDADataset.Options.StrictUpdate` option is set to False, the `RowsAffected` property is not calculated and becomes equal zero. This can improve performance of query executing, so if you need to execute many data updating statements at once and you don't mind affected rows count, set this option to False.

**Fetch**

In some situations you can increase performance a bit by using `TCustomDADataset.Options.CompressBlobMode`.

You can also tweak your application performance by using the following properties of `TCustomDADataset` descendants:

- `FetchRows`
- `Options.FlatBuffers`
- `Options.LongStrings`
- `UniDirectional`

See the descriptions of these properties for more details and recommendations.

**Navigate**

The `Locate` function works faster when dataset is locally sorted on KeyFields fields. Local dataset sorting can be set with the `IndexFieldNames` property. Performance gain can be large if the dataset contains a large number of rows.

Lookup fields work faster when lookup dataset is locally sorted on lookup Keys.

Setting the `TDADatasetOptions.CacheCalcFields` property can improve performance when locally sorting and locating on calculated and lookup fields. It can be also useful when calculated field expressions contain complicated calculations.

Setting the `TDADatasetOptions.LocalMasterDetail` option can improve performance greatly by avoiding server requests on detail refreshes. Setting the `TDADatasetOptions.DetailDelay` option can be useful for avoiding detail refreshes when switching master DataSet records frequently.

**Update**

If your application updates datasets in the CachedUpdates mode, then setting the `TCustomDADataset.Options.UpdateBatchSize` option to more than 1 can improve
performance several hundred times more by reducing the number of requests to the server.

You can also increase the data sending performance a bit (several percents) by using Dataset.UpdateObject.ModifyObject, Dataset.UpdateObject, etc. Little additional performance improvement can be reached by setting the AutoPrepare property for these objects.

**Insert**

If you are about to insert a large number of records into a table, you should use the T:Devart.PgDac.TPgLoader component instead of Insert/Post methods, or execution of the INSERT commands multiple times in a cycle. Sometimes usage of T:Devart.PgDac.TPgLoader improves performance several times.

**4.11 Macros**

Macros help you to change SQL statements dynamically. They allow partial replacement of the query statement by user-defined text. Macros are identified by their names which are then referred from SQL statement to replace their occurrences for associated values.

First step is to assign macros with their names and values to a dataset object.

Then modify SQL statement to include macro names into desired insertion points. Prefix each name with & ("at") sign to let PgDAC discriminate them at parse time. Resolved SQL statement will hold macro values instead of their names but at the right places of their occurrences. For example, having the following statement with the TableName macro name:

```sql
SELECT * FROM &TableName
```

You may later assign any actual table name to the macro value property leaving your SQL statement intact.

```delphi
Query1.SQL.Text := 'SELECT * FROM &TableName';
Query1.MacroByName('TableName').Value := 'Dept';
Query1.Open;
```

PgDAC replaces all macro names with their values and sends SQL statement to the server when SQL execution is requested.

Note that there is a difference between using TMacro AsString and Value properties. If you set macro with the AsString property, it will be quoted. For example, the following statements will result in the same result Query1.SQL property value.

```delphi
Query1.MacroByName('StringMacro').Value := '''A string''';
```
Query1.MacroByName('StringMacro').AsString := 'A string';

Macros can be especially useful in scripts that perform similar operations on different objects. You can use macros that will be replaced with an object name. It allows you to have the same script text and to change only macro values.

You may also consider using macros to construct adaptable conditions in WHERE clauses of your statements.

See Also

- TMacro
- TCustomDADataset.MacroByName
- TCustomDADataset.Macros

© 1997-2021 Devart. All Rights Reserved.

4.12 DataSet Manager

DataSet Manager window

The DataSet Manager window displays the datasets in your project. You can use the DataSet Manager window to create a user interface (consisting of data-bound controls) by dragging items from the window onto forms in your project. Each item has a drop-down control list where you can select the type of control to create prior to dragging it onto a form. You can customize the control list with additional controls, including the controls you have created.
Using the DataSet Manager window, you can:

- Create forms that display data by dragging items from the DataSet Manager window onto forms.

- Customize the list of controls available for each data type in the DataSet Manager window.

- Choose which control should be created when dragging an item onto a form in your Windows application.

- Create and delete TField objects in the DataSets of your project.

**Opening the DataSet Manager window**

You can display the DataSet Manager window by clicking DataSet Manager on the Tools menu. You can also use IDE desktop saving/loading to save DataSet Manager window position and restore it during the next IDE loads.

**Observing project DataSets in the DataSet Manager Window**
By default DataSet Manager shows DataSets of currently open forms. It can also extract DataSets from all forms in the project. To use this, click *Extract DataSets from all forms in project* button. This settings is remembered. Note, that using this mode can slow down opening of the large projects with plenty of forms and DataSets. Opening of such projects can be very slow in Delphi 6 and Borland Developer Studio 2006 and can take up to several tens of minutes.

DataSets can be grouped by form or connection. To change DataSet grouping click the *Grouping mode* button or click a down. You can also change grouping mode by selecting required mode from the DataSet Manager window popup menu.

Creating Data-bound Controls

You can drag an item from the DataSet Manager window onto a form to create a new data-bound control. Each node in the DataSet Manager window allows you to choose the type of control that will be created when you drag it onto a form. You must choose between a Grid layout, where all columns or properties are displayed in a TDataGrid component, or a Details layout, where all columns or properties are displayed in individual controls.

To use grid layout drag the dataset node on the form. By default TDataSource and TDBGrid components are created. You can choose the control to be created prior to dragging by selecting an item in the DataSet Manager window and choosing the control from the item’s drop-down control list.
To use Details layout choose Details from the DataSet node drop-down control list in the
DataSet Manager window. Then select required controls in the drop-down control list for each
DataSet field. DataSet fields must be created. After setting required options you can drag the
DataSet to the form from the DataSet wizard. DataSet Manager will create TDataSource
component, and a component and a label for each field.
Adding custom controls to the DataSet Manager window

To add custom control to the list click the Options button on the DataSet Manager toolbar. A DataSet Manager - Customize controls dialog will appear. Using this dialog you can set controls for the DataSets and for the DataSet fields of different types. To do it, click DataSets node or the node of field of required type in DB objects groups box and use Add and Remove buttons to set required control list. You can also set default control by selecting it in the list of assigned DB controls and pressing Default button.

The default configuration can easily be restored by pressing Reset button in the DataSet Manager - Options dialog.

Working with TField objects

DataSet Manager allows you to create and remove TField objects. DataSet must be active to work with its fields in the DataSet Manager. You can add fields, based on the database table columns, create new fields, remove fields, use drag-n-drop to change fields order.
To create a field based on the database table column right-click the Fields node and select *Create Field* from the popup menu or press `<Insert>`. Note that after you add at least one field manually, DataSet fields corresponding to data fields will not be generated automatically when you drag the DataSet on the form, and you can not drag such fields on the form. To add all available fields right-click the Fields node and select *Add all fields* from the popup menu.

To create new field right-click the Fields node and select *New Field* from the popup menu or press `<Ctrl+Insert>`. The New Field dialog box will appear. Enter required values and press OK button.

To delete fields select these fields in the DataSet Manager window and press `<Delete>`.

DataSet Manager allows you to change view of the fields displayed in the main window. Open the *Customize controls* dialog, and jump to the Options page.

You can choose what information will be added to names of the Field and Data Field objects in the main window of DataSet Manager. Below you can see the example.
There are cases when you need to put large amount of data to a PostgreSQL database. Of course, you may construct INSERT SQL statement and execute it with the TPgSQL component. But it takes a lot of time. PostgreSQL provides COPY FROM STDIN command that allows to load data much faster. PgDAC simplifies using this command by the TPgLoader component.

The COPY command has two modes: text and binary. TPgAlerter supports both these modes. By default the binary mode is used for a connection with 3.0 protocol. In the binary mode TPgLoader works a little faster but some data types are not supported in this mode. Set the TextMode property to True to force text mode. In the text mode you can load data to columns with any PostgreSQL data type.

**Note:** COPY stops operation at the first error. But the target table will already have received earlier rows in a COPY FROM. These rows will not be visible or accessible, but they will still occupy disk space. This might amount to a considerable amount of wasted disk space if the failure happened well into a large copy operation. You may wish to invoke VACUUM to clean the wasted space.

To write your own loader you should:
- create a TPgLoader component;
- set the name of the loading table to TableName;
- create columns which will be loaded (use the TPgLoader component editor at design time);
- write your own event handler: OnGetColumnData or OnPutData;
- call the Load method to start loading.

**See Also**
- TPgLoader
- TPgLoaderColumn
4.14 Large Objects

PostgreSQL has a large object facility which provides stream-style access to user data that is stored in a special large-object structure. Streaming access is useful when working with data values that are too large to manipulate conveniently as a whole.

All large objects are placed in a single system table called pg_largeobject. Each large object has its own OID in this table.

The TPgLargeObject class of PgDAC can be used to create, read, write and delete large objects. To manipulate with large objects create an instance of TPgLargeObject and specify the connection that will be used for operations with a large object. If you are working with an existent large object, specify its OID.

Creating a new object:

```pascal
var
  LargeObject: TPgLargeObject;
  AData: array [1..10] of byte;
...
  PgConnection.StartTransaction;
  LargeObject := TPgLargeObject.Create(PgConnection);
  try
    LargeObject.CreateObject;
    LargeObject.WriteBlob(0, 10, &AData);
    LargeObject.CloseObject;
  finally
    LargeObject.Free;
  end;
  PgConnection.Commit;
```

Reading an existent object:

```pascal
...
  LargeObject := TPgLargeObject.Create(PgConnection);
  try
    LargeObject.OID := 12345;
    LargeObject.OpenObject;
    LargeObject.Read(0, 10, &AData);
    LargeObject.CloseObject;
  finally
    LargeObject.Free;
  end;
...
```

Note that manipulations with large objects require a transaction. So StartTransaction is called in the example.

By default TPgLargeObject instance uses a memory buffer to hold a value of large object. On the first call to the Read method the TPgLargeObject reads the whole object value and stores
it in the buffer. You can also call the ReadBlob method to read a value to the buffer.

When you write data using the Write method, the data are stored in memory buffer. You should call the WriteBlob method to pass the data to the database.

When working with very large objects, you can set the Cached property to False. In this case the memory buffer is not used, and the Read and Write methods work directly with a value in the database.

If you open a table with a column of the OID data type, TCustomPgDataSet descendant components assume that values in this column are large objects OIDs, and automatically read data from the corresponding large objects.

If OIDs are not large objects OIDs, set the OIDAsInt option of TCustomPgDataSet to True. In this case OID columns are read as simple integer columns.

You can use the DefferedBlobRead and CacheBlobs options to optimize performance and memory usage. If you set the DefferedBlobRead option to True, the dataset does not read large object data when it fetches records. When you access a value of a large object field, the data for the corresponding large object have been read.

If you set the CacheBlobs option to False, all large objects in the dataset do not cache their values. In this case the DefferedBlobRead value has no sense.

© 1997-2021 Devart. All Rights Reserved.

4.15 REFCURSOR Data Type

Rather than executing a whole query at once, it is possible to set up a cursor that encapsulates the query, and then read the query result a few rows at a time. One reason for doing this is to avoid memory overrun when the result contains a large number of rows.

PL/pgSQL functions can return cursors to the caller. This is useful to return multiple rows or columns, especially with very large result sets. To do this, the function opens the cursor and returns the cursor name to the caller (or simply opens the cursor using a portal name specified by or otherwise known to the caller). The caller can then fetch rows from the cursor. The cursor can be closed by the caller, or it will be closed automatically when the transaction closes.

PgDAC supports reading cursors returned from stored procedures. The TPgStoredProc component opens automatically the first REFCURSOR returned from a stored procedure. For example, consider the following procedure:
CREATE FUNCTION cursor_func() RETURNS REFCURSOR AS $$
DECLARE
    ref REFCURSOR;
BEGIN
    OPEN ref FOR SELECT * FROM test;
    RETURN ref;
END;
$$ LANGUAGE plpgsql;

You can read data from the returned cursor using the following code:

PgConnection.StartTransaction;
PgStoredProc.StoredProcName := 'cursor_func';
PgStoredProc.Open;
while not PgStoredProc.Eof do begin
    Value := PgStoredProc.Fields[0].AsInteger;
    ...
    PgStoredProc.Next;
end;
PgStoredProc.Close;
PgConnection.Commit;

Note that using cursors requires a transaction. So that StartTransaction is called before the Open method of TPgStoredProc.

If a stored procedure returns several REFCURSOR parameters, only first cursor is opened when you call the Open method of TPgStoredProc. To open the rest of cursors you can use the OpenNext method, or manipulate with TPgRefCursor instances. For example:

CREATE FUNCTION cursor_func(c1 INOUT REFCURSOR, c2 INOUT REFCURSOR) RETURNS RECORD AS $$
BEGIN
    OPEN c1 FOR SELECT * FROM test1;
    OPEN c2 FOR SELECT * FROM test2;
END;
$$ LANGUAGE plpgsql;

You can read data using the following code:

PgConnection.StartTransaction;
PgStoredProc.StoredProcName := 'cursor_func';
PgStoredProc.Open;
repeat
    while not PgStoredProc.Eof do begin
        Value := PgStoredProc.Fields[0].AsInteger;
        ...
        PgStoredProc.Next;
    end;
until not PgStoredProc.OpenNext;
PgStoredProc.Close;
PgConnection.Commit;

You can open both cursors at the same time by assigning a TPgRefCursor instance to the Cursor property of a dataset:

var
4.16 National and Unicode Characters

On transferring data between client and server sides, server must know the encoding used at the client. By default client encoding is the same as the database encoding. You can set the encoding using `TPgConnection.Options_CHARSET` or `TPgConnection.Options.UseUnicode` properties. The Charset and UseUnicode options are mutually exclusive, thus on setting the UseUnicode property to True a value of Charset will be ignored.

If the Charset property is set, then on establishing a connection "SET client_encoding = <Charset>" query is automatically passed to the server to explicitly notify the server about the character set of the client. Pay attention that on setting Charset to UTF8 values of all string fields will be converted to this encoding that in most cases can make impossible to use DataAware components.

When you set the UseUnicode option to True, PgDAC also uses UTF8 encoding but it automatically converts all string values to Unicode (UTF-16). `TWideStringField` and `TWideMemoField` field types are used instead of `TStringField` and `TMemoField`. Setting the UseUnicode option to True lets you work simultaneously almost with all languages. This behaviour is suitable, for example, when creating a database of books in the library, when next to the title of a book you should also store its title in the original language.

See Also

- `TPgConnection.Options`
4.17 **Connection Pooling**

Connection pooling enables an application to use a connection from a pool of connections that do not need to be reestablished for each use. Once a connection has been created and placed in a pool, an application can reuse that connection without performing the complete connection process.

Using a pooled connection can result in significant performance gains, because applications can save the overhead involved in making a connection. This can be particularly significant for middle-tier applications that connect over a network or for applications that connect and disconnect repeatedly, such as Internet applications.

To use connection pooling set the Pooling property of the TCustomDACConnection component to True. Also you should set the PoolingOptions of the TCustomDACConnection. These options include MinPoolSize, MaxPoolSize, Validate, ConnectionLifeTime. Connections belong to the same pool if they have identical values for the following parameters: MinPoolSize, MaxPoolSize, Validate, ConnectionLifeTime, Server, Username, Password, Database, Port, ProtocolVersion, Charset, UseUnicode, Schema, ConnectionTimeout, SSLOptions. When a connection component disconnects from the database the connection actually remains active and is placed into the pool. When this or another connection component connects to the database it takes a connection from the pool. Only when there are no connections in the pool, new connection is established.

Connections in the pool are validated to make sure that a broken connection will not be returned for the TCustomDACConnection component when it connects to the database. The pool validates connection when it is placed to the pool (e.g. when the TCustomDACConnection component disconnects). If connection is broken it is not placed to the pool. Instead the pool frees this connection. Connections that are held in the pool are validated every 30 seconds. All broken connections are freed. If you set the PoolingOptions.Validate to True, a connection also will be validated when the TCustomDACConnection component connects and takes a connection from the pool. When some network problem occurs all connections to the database can be broken. Therefore the pool validates all connections before any of them will be used by a TCustomDACConnection component if a fatal error is detected on one connection.

The pool frees connections that are held in the pool during a long time. If no new connections are placed to the pool it becomes empty after approximately 4 minutes. This pool behaviour is intended to save resources when the count of connections in the pool exceeds the count that is needed by application. If you set the PoolingOptions.MinPoolSize property to a non-zero value, this prevents the pool from freeing all pooled connections. When connection count in the pool decreases to MinPoolSize value, remaining connection will not be freed except if they
are broken.

The `PoolingOptions.MaxPoolSize` property limits the count of connections that can be active at the same time. If maximum count of connections is active and some `TCustomDAConnection` component tries to connect, it will have to wait until any of `TCustomDAConnection` components disconnect. Maximum wait time is 30 seconds. If active connections' count does not decrease during 30 seconds, the `TCustomDAConnection` component will not connect and an exception will be raised.

You can limit the time of connection's existence by setting the `PoolingOptions.ConnectionLifeTime` property. When the `TCustomDAConnection` component disconnects, its internal connection will be freed instead of placing to the pool if this connection is active during the time longer than the value of the `PoolingOptions.ConnectionLifeTime` property. This property is designed to make load balancing work with the connection pool.

To force freeing of a connection when the `TCustomDAConnection` component disconnects, the `RemoveFromPool` method of `TCustomDAConnection` can be used. You can also free all connection in the pool by using the class procedures `Clear` or `AsyncClear` of `TPgConnectionPoolManager`. These procedures can be useful when you know that all connections will be broken for some reason.

It is recommended to use connection pooling with the `DisconnectMode` option of the `TCustomDAConnection` component set to True. In this case internal connections can be shared between `TCustomDAConnection` components. When some operation is performed on the `TCustomDAConnection` component (for example, an execution of SQL statement) this component will connect using pooled connection and after performing operation it will disconnect. When an operation is performed on another `TCustomDAConnection` component it can use the same connection from the pool.

See Also

- `TCustomDAConnection.Pooling`
- `TCustomDAConnection.PoolingOptions`
- `Working with Disconnected Mode`
4.18 **DBMonitor**

To extend monitoring capabilities of PgDAC applications there is an additional tool called DBMonitor. It is provided as an alternative to Borland SQL Monitor which is also supported by PgDAC.

DBMonitor is an easy-to-use tool to provide visual monitoring of your database applications.

DBMonitor has the following features:
- multiple client processes tracing;
- SQL event filtering (by sender objects);
- SQL parameter and error tracing.

DBMonitor is intended to hamper an application being monitored as little as possible.

To trace your application with DB Monitor you should follow these steps:
- drop **TPgSQLMonitor** component onto the form;
- turn **moDBMonitor** option on;
- set to True the Debug property for components you want to trace;
- start DBMonitor before running your program.

© 1997-2021 Devart. All Rights Reserved.

---

4.19 **Writing GUI Applications with PgDAC**

PgDAC GUI part is standalone. This means that to make GUI elements such as SQL cursors, connect form, connect dialog etc. available, you should explicitly include PgDacVcl unit in your application. This feature is needed for writing console applications.

**Delphi and C++Builder**

By default PgDAC does not require Forms, Controls and other GUI related units. Only **TPgConnectDialog** component require the Forms unit.

© 1997-2021 Devart. All Rights Reserved.

---

4.20 **Compatibility with Previous Versions**

We always try to keep PgDAC compatible with previous versions, but sometimes we have to change the behaviour of PgDAC in order to enhance its functionality, or avoid bugs. This topic
describes such changes, and how to revert the old PgDAC behaviour. We strongly recommend not to turn on the old behaviour of PgDAC. Use options described below only if changes applied to PgDAC crashed your existent application.

Values of the options described below should be assigned in the initialization section of one of the units in your project.

**DBAccess.BaseSQLOldBehavior:**

The `BaseSQL` property is similar to the SQL property, but it does not store changes made by `AddWhere`, `DeleteWhere`, and `SetOrderBy` methods. After assigning an SQL text and modifying it by one of these methods, all subsequent changes of the SQL property will not be reflected in the `BaseSQL` property. This behavior was changed in PgDAC. To restore old behavior, set the `BaseSQLOldBehavior` variable to True.

**DBAccess.SQLGeneratorCompatibility:**

If the manually assigned `RefreshSQL` property contains only "WHERE" clause, PgDAC uses the value of the `BaseSQL` property to complete the refresh SQL statement. In this situation all modifications applied to the SELECT query by functions `AddWhere`, `DeleteWhere` are not taken into account. This behavior was changed in PgDAC. To restore the old behavior, set the `BaseSQLOldBehavior` variable to True.

**MemDS.SendDataSetChangeEventAfterOpen:**

Starting with PgDAC, the DataSetChangeEvent is sent after the dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. This problem appears only under Windows XP when visual styles are enabled.

To disable sending this event, change the value of this variable to False.

**MemDS.DoNotRaiseExceptionOnUaFail:**

Starting with PgDAC, if the `OnUpdateRecord` event handler sets the UpdateAction parameter to uaFail, an exception is raised. The default value of UpdateAction is uaFail. So, the exception will be raised when the value of this parameter is left unchanged.

To restore the old behaviour, set `DoNotRaiseExceptionOnUaFail` to True.
4.21 64-bit Development with Embarcadero RAD Studio XE2

RAD Studio XE2 Overview

RAD Studio XE2 is the major breakthrough in the line of all Delphi versions of this product. It allows deploying your applications both on Windows and Mac OS platforms. Additionally, it is now possible to create 64-bit Windows applications to fully benefit from the power of new hardware. Moreover, you can create visually spectacular applications with the help of the FireMonkey GPU application platform.

Its main features are the following:
- Windows 64-bit platform support;
- Mac OS support;
- FireMonkey application development platform;
- Live data bindings with visual components;
- VCL styles for Windows applications.

Changes in 64-bit Application Development

64-bit platform support implies several important changes that each developer must keep in mind prior to the development of a new application or the modernization of an old one.

General

RAD Studio XE2 IDE is a 32-bit application. It means that it cannot load 64-bit packages at design-time. So, all design-time packages in RAD Studio XE2 IDE are 32-bit.

Therefore, if you develop your own components, you should remember that for the purpose of developing components with the 64-bit platform support, you have to compile run-time packages both for the 32- and 64-bit platforms, while design-time packages need to be compiled only for the 32-bit platform. This might be a source of difficulties if your package is simultaneously both a run-time and a design-time package, as it is more than likely that this package won't be compiled for the 64-bit platform. In this case, you will have to separate your package into two packages, one of which will be used as run-time only, and the other as design-time only.

For the same reason, if your design-time packages require that certain DLLs be loaded, you should remember that design-time packages can be only 32-bit and that is why they can load only 32-bit versions of these DLLs, while at run-time 64-bit versions of the DLLs will be loaded. Correspondingly, if there are only 64-bit versions of the DLL on your computer, you won't be able to use all functions at design-time and, vice versa, if you have only 32-bit...
versions of the DLLs, your application won't be able to work at run-time.

**Extended type**

For this type in a 64-bit applications compiler generates SSE2 instructions instead of FPU, and that greatly improves performance in applications that use this type a lot (where data accuracy is needed). For this purpose, the size and precision of Extended type is reduced:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended</td>
<td>10 bytes</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

The following two additional types are introduced to ensure compatibility in the process of developing 32- and 64-bit applications:

Extended80 – whose size in 32-bit application is 10 bytes; however, this type provides the same precision as its 8-byte equivalent in 64-bit applications.

Extended80Rec – can be used to perform low-level operations on an extended precision floating-point value. For example, the sign, the exponent, and the mantissa can be changed separately. It enables you to perform memory-related operations with 10-bit floating-point variables, but not extended-precision arithmetic operations.

**Pointer and Integers**

The major difference between 32- and 64-bit platforms is the volume of the used memory and, correspondingly, the size of the pointer that is used to address large memory volumes.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer</td>
<td>4 bytes</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

At the same time, the size of the Integer type remains the same for both platforms:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>4 bytes</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

That is why, the following code will work incorrectly on the 64-bit platform:

```
Ptr := Pointer(Integer(Ptr) + Offset);
```

While this code will correctly on the 64-bit platform and incorrectly on the 32-bit platform:

```
Ptr := Pointer(Int64(Ptr) + Offset);
```

For this purpose, the following platform-dependent integer type is introduced:
**TYPE** | **32-bit** | **64-bit**
---|---|---
NativeInt | 4 bytes | 8 bytes
NativeUInt | 4 bytes | 8 bytes

This type helps ensure that pointers work correctly both for the 32- and 64-bit platforms:

```pascal
Ptr := Pointer(NativeInt(Ptr) + Offset);
```

However, you need to be extra-careful when developing applications for several versions of Delphi, in which case you should remember that in the previous versions of Delphi the `NativeInt` type had different sizes:

<table>
<thead>
<tr>
<th><strong>TYPE</strong></th>
<th><strong>Delphi Version</strong></th>
<th><strong>Size</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>NativeInt</td>
<td>D5</td>
<td>N/A</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D6</td>
<td>N/A</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D7</td>
<td>8 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D2005</td>
<td>8 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D2006</td>
<td>8 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D2007</td>
<td>8 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D2009</td>
<td>4 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>D2010</td>
<td>4 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>Delphi XE</td>
<td>4 bytes</td>
</tr>
<tr>
<td>NativeInt</td>
<td>Delphi XE2</td>
<td>4 or 8 bytes</td>
</tr>
</tbody>
</table>

**Out parameters**

Some WinAPIs have OUT parameters of the `SIZE_T` type, which is equivalent to `NativeInt` in Delphi XE2. The problem is that if you are developing only a 32-bit application, you won't be able to pass `Integer` to OUT, while in a 64-bit application, you will not be able to pass `Int64`; in both cases you will have to pass `NativeInt`.

For example:

```pascal
procedure MyProc(out Value: NativeInt);
begin
  Value := 12345;
end;
var
  Value1: NativeInt;
  {$IFDEF WIN32}
  Value2: Integer;
  {$ENDIF}
  {$IFDEF WIN64}
  Value2: Int64;
```
{ENDIF}
begin   MyProc(Value1); // will be compiled;
       MyProc(Value2); // will not be compiled !!!
end;

Win API

If you pass pointers to SendMessage/PostMessage/TControl.Perform, the wParam and
lParam parameters should be type-casted to the WPARAM/LPARAM type and not to Integer/
Longint.

Correct:

SendMessage(hWnd, WM_SETTEXT, 0, LPARAM(@MyCharArray));

Wrong:

SendMessage(hWnd, WM_SETTEXT, 0, Integer(@MyCharArray));

Replace SetWindowLong/GetWindowLog with SetWindowLongPtr/GetWindowLongPtr for
GWLP_HINSTANCE, GWLP_ID, GWLP_USERDATA, GWLP_HWNDPARENT and
GWLP_WNDPROC as they return pointers and handles. Pointers that are passed to
SetWindowLongPtr should be type-casted to LONG_PTR and not to Integer/Longint.

Correct:

SetWindowLongPtr(hWnd, GWLP_WNDPROC, LONG_PTR(@MyWindowProc));

Wrong:

SetWindowLong(hWnd, GWL_WNDPROC, Longint(@MyWindowProc));

Pointers that are assigned to the TMessage.Result field should use a type-cast to LRESULT
instead of Integer/Longint.

Correct:

Message.Result := LRESULT(Self);

Wrong:

Message.Result := Integer(Self);

All TWM...-records for the windows message handlers must use the correct Windows types
for the fields:

Msg: UINT; wParam: WPARAM; lParam: LPARAM; Result: LRESULT

Assembler

In order to make your application (that uses assembly code) work, you will have to make
several changes to it:
Using PgDAC

- rewrite your code that mixes Pascal code and assembly code. Mixing them is not supported in 64-bit applications;
- rewrite assembly code that doesn't consider architecture and processor specifics.

You can use conditional defines to make your application work with different architectures.


### Exception handling

The biggest difference in exception handling between Delphi 32 and 64-bit is that in Delphi XE2 64-bit you will gain more performance because of different internal exception mechanism. For 32-bit applications, the Delphi compiler (dcc32.exe) generates additional code that is executed any way and that causes performance loss. The 64-bit compiler (dcc64.exe) doesn't generate such code, it generates metadata and stores it in the PDATA section of an executable file instead.

But in Delphi XE2 64-bit it's impossible to have more than 16 levels of nested exceptions. Having more than 16 levels of nested exceptions will cause a Run Time error.

### Debugging

Debugging of 64-bit applications in RAD Studio XE2 is remote. It is caused by the same reason: RAD Studio XE2 IDE is a 32 application, but your application is 64-bit. If you are trying to debug your application and you cannot do it, you should check that the Include remote debug symbols project option is enabled.

To enable it, perform the following steps:

1. Open Project Options (in the main menu Project->Options).
2. In the Target combobox, select Debug configuration - 64-bit Windows platform. If there is no such option in the combobox, right click "Target Platforms" in Project Manager and select Add platform. After adding the 64-bit Windows platform, the Debug configuration - 64-bit Windows platform option will be available in the Target combobox.
3. Select Linking in the left part of the Project Options form.
4. enable the Include remote debug symbols option.

After that, you can run and debug your 64-bit application.

To enable remote debugging, perform the following steps:

1. Install Platform Assistant Server (PAServer) on a remote computer. You can find PAServer
in the %RAD_Studio_XE2_Install_Directory%\PAServer directory. The setup_paserver.exe file is an installation file for Windows, and the setup_paserver.zip file is an installation file for MacOS.

2. Run the PAServer.exe file on a remote computer and set the password that will be used to connect to this computer.

3. On a local computer with RAD Studio XE2 installed, right-click the target platform that you want to debug in Project Manager and select Assign Remote Profile. Click the Add button in the displayed window, input your profile name, click the Next button, input the name of a remote computer and the password to it (that you assigned when you started PAServer on a remote computer).

After that, you can test the connection by clicking the Test Connection button. If your connection failed, check that your firewalls on both remote and local computers do not block your connection, and try to establish a connection once more. If your connection succeeded, click the Next button and then the Finish button. Select your newly created profile and click OK.

After performing these steps you will be able to debug your application on a remote computer. Your application will be executed on a remote computer, but you will be able to debug it on your local computer with RAD Studio XE2.

For more information about working with Platform Assistant Server, please refer to [http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Running_the_Platform_Assistant_on_Windows](http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Running_the_Platform_Assistant_on_Windows)

4.22 Database Specific Aspects of 64-bit Development

**PostgreSQL Connectivity Aspects**

Since PgDAC does not require that the PostgreSQL client be installed to work with the database, the development of applications for the x64 platform does not differ from the development of application for Windows x86.
5 Reference

This page shortly describes units that exist in PgDAC.

Units

<table>
<thead>
<tr>
<th>Unit Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRAccess</td>
<td>This unit contains base classes for accessing databases.</td>
</tr>
<tr>
<td>CRBatchMove</td>
<td>This unit contains implementation of the TCRBatchMove component.</td>
</tr>
<tr>
<td>CREncryption</td>
<td>This unit contains base classes for data encryption.</td>
</tr>
<tr>
<td>CRGrid</td>
<td>This unit contains the TCRDBGrid component.</td>
</tr>
<tr>
<td>CRVio</td>
<td>This unit contains classes for HTTP connections.</td>
</tr>
<tr>
<td>DAAlerter</td>
<td>This unit contains the base class for the TPgAlerter component.</td>
</tr>
<tr>
<td>DADump</td>
<td>This unit contains the base class for the TPgDump component.</td>
</tr>
<tr>
<td>DALoader</td>
<td>This unit contains the base class for the TPgLoader component.</td>
</tr>
<tr>
<td>DAScript</td>
<td>This unit contains the base class for the TPgScript component.</td>
</tr>
<tr>
<td>DASQLMonitor</td>
<td>This unit contains the base class for the TPgSQLMonitor component.</td>
</tr>
<tr>
<td>DBAccess</td>
<td>This unit contains base classes for most of the components.</td>
</tr>
<tr>
<td>Devart.Dac.DataAdapter</td>
<td>This unit contains implementation of the DADataAdapter class.</td>
</tr>
<tr>
<td>Devart.PgDac.DataAdapter</td>
<td>This unit contains implementation of the PgDataAdapter class.</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>MemData</strong></td>
<td>This unit contains classes for storing data in memory.</td>
</tr>
<tr>
<td><strong>MemDS</strong></td>
<td>This unit contains implementation of the TMemDataSet class.</td>
</tr>
<tr>
<td><strong>PgAccess</strong></td>
<td>This unit contains main components of PgDAC.</td>
</tr>
<tr>
<td><strong>PgAlerter</strong></td>
<td>This unit contains the implementation of the TPgAlerter component.</td>
</tr>
<tr>
<td><strong>PgClasses</strong></td>
<td>This unit contains the implementation of internal PgDAC classes and types.</td>
</tr>
<tr>
<td><strong>PgConnectionPool</strong></td>
<td>This unit contains the TPgConnectionPoolManager class for managing connection pool.</td>
</tr>
<tr>
<td><strong>PgDacVcl</strong></td>
<td>This unit contains the visual constituent of PgDAC.</td>
</tr>
<tr>
<td><strong>PgDump</strong></td>
<td>This unit contains the implementation of the TPgDump component.</td>
</tr>
<tr>
<td><strong>PgError</strong></td>
<td>This unit contains the EPgError exception class.</td>
</tr>
<tr>
<td><strong>PgLoader</strong></td>
<td>This unit contains the implementation of the TPgLoader component.</td>
</tr>
<tr>
<td><strong>PgObjects</strong></td>
<td>This unit contains classes for PostgreSQL specific data types.</td>
</tr>
<tr>
<td><strong>PgScript</strong></td>
<td>This unit contains the implementation of the TPgScript component.</td>
</tr>
<tr>
<td><strong>PgSQLMonitor</strong></td>
<td>This unit contains the implementation of the TPgSQLMonitor component.</td>
</tr>
<tr>
<td><strong>VirtualDataSet</strong></td>
<td>This unit contains implementation of the TVirtualDataSet component.</td>
</tr>
<tr>
<td><strong>VirtualTable</strong></td>
<td>This unit contains implementation of the TVirtualTable component.</td>
</tr>
</tbody>
</table>
5.1 **CRAccess**

This unit contains base classes for accessing databases.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRCursor</td>
<td>A base class for classes that work with database cursors.</td>
</tr>
</tbody>
</table>

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBeforeFetchProc</td>
<td>This type is used for the TCustomDADataSet.Before Fetch event.</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRIsolationLevel</td>
<td>Specifies how to handle transactions containing database modifications.</td>
</tr>
<tr>
<td>TCRTransactionAction</td>
<td>Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.</td>
</tr>
<tr>
<td>TCursorState</td>
<td>Used to set cursor state</td>
</tr>
</tbody>
</table>

5.1.1 **Classes**

Classes in the **CRAccess** unit.
5.1.1.1 TCRCursor Class

A base class for classes that work with database cursors.

For a list of all members of this type, see TCRCursor members.

Unit

CRAccess

Syntax

TCRCursor = class(TSharedObject);

Remarks

TCRCursor is a base class for classes that work with database cursors.

Inheritance Hierarchy

TSharedObject

TCRCursor

5.1.1.1.1 Members

TCRCursor class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddRef</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td><strong>Release</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback

### 5.1.2 Types

Types in the **CRAccess** unit.

## Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TBeforeFetchProc</strong></td>
<td>This type is used for the TCustomDADataset.BeforeFetch event.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback

#### 5.1.2.1 TBeforeFetchProc Procedure Reference

This type is used for the TCustomDADataset.BeforeFetch event.

### Unit

**CRAccess**

### Syntax

```delphi
TBeforeFetchProc = procedure (var Cancel: boolean) of object;
```

### Parameters

**Cancel**

True, if the current fetch operation should be aborted.

© 1997-2021  
Request Support  
DAC Forum  
Provide Feedback

© 2021 Devart
Devart. All Rights Reserved.

5.1.3 Enumerations

Enumerations in the CRAccess unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRIsolationLevel</td>
<td>Specifies how to handle transactions containing database modifications.</td>
</tr>
<tr>
<td>TCRTransactionAction</td>
<td>Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.</td>
</tr>
<tr>
<td>TCursorState</td>
<td>Used to set cursor state</td>
</tr>
</tbody>
</table>

5.1.3.1 TCRIsolationLevel Enumeration

Specifies how to handle transactions containing database modifications.

Unit

CRAccess

Syntax

```
TCRIsolationLevel = (ilReadCommitted, ilReadUncommitted, ilRepeatableRead, ilIsolated, ilSnapshot, ilCustom);
```

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ilCustom</td>
<td>The parameters of the transaction are set manually in the Params property.</td>
</tr>
<tr>
<td>illsolated</td>
<td>The most restricted level of transaction isolation. Database server isolates data involved in current transaction by putting</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback
additional processing on range locks. Used to put aside all undesired effects observed in the concurrent accesses to the same set of data, but may lead to a greater latency at times of a congested database environment.

**ilReadCommitted**
Sets isolation level at which transaction cannot see changes made by outside transactions until they are committed. Only dirty reads (changes made by uncommitted transactions) are eliminated by this state of the isolation level. The default value.

**ilReadUncommitted**
The most unrestricted level of the transaction isolation. All types of data access interferences are possible. Mainly used for browsing database and to receive instant data with prospective changes.

**ilRepeatableRead**
Prevents concurrent transactions from modifying data in the current uncommitted transaction. This level eliminates dirty reads as well as nonrepeatable reads (repeatable reads of the same data in one transaction before and after outside transactions may have started and committed).

**ilSnapshot**
Uses row versioning. Provides transaction-level read consistency. A data snapshot is taken when the snapshot transaction starts, and remains consistent for the duration of a transaction.

---

### 5.1.3.2 TCRTransactionAction Enumeration

Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

**Unit**

**CRAccess**

**Syntax**

```plaintext
TCRTransactionAction = (taCommit, taRollback);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>taCommit</td>
<td>Transaction is committed.</td>
</tr>
<tr>
<td>taRollback</td>
<td>Transaction is rolled back.</td>
</tr>
</tbody>
</table>
5.1.3.3 TCursorState Enumeration

Used to set cursor state

Unit

CRAccess

Syntax

TCursorState = (csInactive, csOpen, csParsed, csPrepared, csBound, csExecuteFetchAll, csExecuting, csExecuted, csFetching, csFetchingAll, csFetched);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>csBound</td>
<td>Parameters bound</td>
</tr>
<tr>
<td>csExecuted</td>
<td>Statement successfully executed</td>
</tr>
<tr>
<td>csExecuteFetchAll</td>
<td>Set before FetchAll</td>
</tr>
<tr>
<td>csExecuting</td>
<td>Statement is set before executing</td>
</tr>
<tr>
<td>csFetched</td>
<td>Fetch finished or canceled</td>
</tr>
<tr>
<td>csFetching</td>
<td>Set on first</td>
</tr>
<tr>
<td>csFetchingAll</td>
<td>Set on the FetchAll start</td>
</tr>
<tr>
<td>csInactive</td>
<td>Default state</td>
</tr>
<tr>
<td>csOpen</td>
<td>statement open</td>
</tr>
<tr>
<td>csParsed</td>
<td>Statement parsed</td>
</tr>
<tr>
<td>csPrepared</td>
<td>Statement prepared</td>
</tr>
</tbody>
</table>

5.2 CRBatchMove

This unit contains implementation of the TCRBatchMove component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
TCRBatchMove transfers records between datasets.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMoveProgressEvent</td>
<td>This type is used for the TCRBatchMove.OnBatchMoveProgress event.</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMode</td>
<td>Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.</td>
</tr>
<tr>
<td>TCRFieldMappingMode</td>
<td>Used to specify the way fields of the destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.</td>
</tr>
</tbody>
</table>

5.2.1 Classes

Classes in the CRBatchMove unit.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMove</td>
<td>Transfers records between datasets.</td>
</tr>
</tbody>
</table>
5.2.1.1 TCRBatchMove Class

Transfers records between datasets.

For a list of all members of this type, see TCRBatchMove members.

Unit

CRBatchMove

Syntax

TCRBatchMove = class(TComponent);

Remarks

The TCRBatchMove component transfers records between datasets. Use it to copy dataset records to another dataset or to delete datasets records that match records in another dataset. The TCRBatchMove.Mode property determines the desired operation type, the TCRBatchMove.Source and TCRBatchMove.Destination properties indicate corresponding datasets.

Note: A TCRBatchMove component is added to the Data Access page of the component palette, not to the PgDAC page.

© 1997-2021
Devart. All Rights Reserved.

5.2.1.1.1 Members

TCRBatchMove class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbortOnKeyViol</td>
<td>Used to specify whether the batch operation should be terminated immediately after key or integrity violation.</td>
</tr>
<tr>
<td>AbortOnProblem</td>
<td>Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.</td>
</tr>
<tr>
<td><strong>ChangedCount</strong></td>
<td>Used to get the number of records changed in the destination dataset.</td>
</tr>
<tr>
<td><strong>CommitCount</strong></td>
<td>Used to set the number of records to be batch moved before commit occurs.</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>Used to specify the destination dataset for the batch operation.</td>
</tr>
<tr>
<td><strong>FieldMappingMode</strong></td>
<td>Used to specify the way fields of destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.</td>
</tr>
<tr>
<td><strong>KeyViolCount</strong></td>
<td>Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.</td>
</tr>
<tr>
<td><strong>Mappings</strong></td>
<td>Used to set field matching between source and destination datasets for the batch operation.</td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td>Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.</td>
</tr>
<tr>
<td><strong>MovedCount</strong></td>
<td>Used to get the number of records that were read from the source dataset during the batch operation.</td>
</tr>
<tr>
<td><strong>ProblemCount</strong></td>
<td>Used to get the number of records that could not be added to the destination dataset because of the field type mismatch.</td>
</tr>
<tr>
<td><strong>RecordCount</strong></td>
<td>Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Used to specify the source dataset for the batch operation.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Performs the batch operation.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnBatchMoveProgress</td>
<td>Occurs when providing feedback to the user about the batch operation in progress is needed.</td>
</tr>
</tbody>
</table>

Properties of the TCRBatchMove class.

For a complete list of the TCRBatchMove class members, see the TCRBatchMove Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangedCount</td>
<td>Used to get the number of records changed in the destination dataset.</td>
</tr>
<tr>
<td>KeyViolCount</td>
<td>Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.</td>
</tr>
<tr>
<td>MovedCount</td>
<td>Used to get the number of records that were read from the source dataset during the batch operation.</td>
</tr>
<tr>
<td>ProblemCount</td>
<td>Used to get the number of records that could not be</td>
</tr>
</tbody>
</table>
added to the destination dataset because of the field type mismatch.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbortOnKeyViol</td>
<td>Used to specify whether the batch operation should be terminated immediately after key or integrity violation.</td>
</tr>
<tr>
<td>AbortOnProblem</td>
<td>Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.</td>
</tr>
<tr>
<td>CommitCount</td>
<td>Used to set the number of records to be batch moved before commit occurs.</td>
</tr>
<tr>
<td>Destination</td>
<td>Used to specify the destination dataset for the batch operation.</td>
</tr>
<tr>
<td>FieldMappingMode</td>
<td>Used to specify the way fields of destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.</td>
</tr>
<tr>
<td>Mappings</td>
<td>Used to set field matching between source and destination datasets for the batch operation.</td>
</tr>
<tr>
<td>Mode</td>
<td>Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.</td>
</tr>
<tr>
<td>RecordCount</td>
<td>Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.</td>
</tr>
<tr>
<td>Source</td>
<td>Used to specify the source</td>
</tr>
</tbody>
</table>
See Also

- TCRBatchMove Class
- TCRBatchMove Class Members

5.2.1.1.2.1  AbortOnKeyViol Property

Used to specify whether the batch operation should be terminated immediately after key or integrity violation.

Class

TCRBatchMove

Syntax

```pascal
property AbortOnKeyViol: boolean default True;
```

Remarks

Use the AbortOnKeyViol property to specify whether the batch operation is terminated immediately after key or integrity violation.

5.2.1.1.2.2  AbortOnProblem Property

Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.

Class

TCRBatchMove

Syntax

```pascal
property AbortOnProblem: boolean default True;
```
Remarks

Use the AbortOnProblem property to specify whether the batch operation is terminated immediately when it is necessary to truncate data to make it fit the specified Destination.

5.2.1.2.3 ChangedCount Property

Used to get the number of records changed in the destination dataset.

Class

TCRBatchMove

Syntax

```delphi
property ChangedCount: Integer;
```

Remarks

Use the ChangedCount property to get the number of records changed in the destination dataset. It shows the number of records that were updated in the bmUpdate or bmAppendUpdate mode or were deleted in the bmDelete mode.

5.2.1.2.4 CommitCount Property

Used to set the number of records to be batch moved before commit occurs.

Class

TCRBatchMove

Syntax

```delphi
property CommitCount: integer default 0;
```

Remarks

Use the CommitCount property to set the number of records to be batch moved before the commit occurs. If it is set to 0, the operation will be chunked to the number of records to fit 32
5.2.1.1.2.5 Destination Property

Used to specify the destination dataset for the batch operation.

Class

TCRBatchMove

Syntax

```
property Destination: TDataSet;
```

Remarks

Specifies the destination dataset for the batch operation.

5.2.1.1.2.6 FieldMappingMode Property

Used to specify the way fields of destination and source datasets will be mapped to each other if the Mappings list is empty.

Class

TCRBatchMove

Syntax

```
property FieldMappingMode: TCRFieldMappingMode default mmFieldIndex;
```

Remarks

Specifies in what way fields of destination and source datasets will be mapped to each other if the Mappings list is empty.
5.2.1.1.2.7 KeyViolCount Property

Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.

Class

TCRBatchMove

Syntax

\[
\text{property KeyViolCount: Integer;}
\]

Remarks

Use the KeyViolCount property to get the number of records that could not be replaced, added, deleted from the destination dataset because of integrity or key violations.

If \text{AbortOnKeyViol} is True, then KeyViolCount will never exceed one, because the operation aborts when the integrity or key violation occurs.

See Also

- \text{AbortOnKeyViol}

© 1997-2021
Devart. All Rights Reserved.

5.2.1.1.2.8 Mappings Property

Used to set field matching between source and destination datasets for the batch operation.

Class

TCRBatchMove

Syntax

\[
\text{property Mappings: TStrings;}
\]

Remarks

Use the Mappings property to set field matching between the source and destination datasets for the batch operation. By default fields matching is based on their position in the datasets. To map the column ColName in the source dataset to the column with the same name in the destination dataset, use:
Example

To map a column named SourceColName in the source dataset to the column named DestColName in the destination dataset, use:

```
DestColName=SourceColName
```

5.2.1.2.9  Mode Property

Used to set the type of the batch operation that will be executed after calling the `Execute` method.

Class

`TCRBatchMove`

Syntax

```java
property Mode: TCRBatchMode default bmAppend;
```

Remarks

Use the Mode property to set the type of the batch operation that will be executed after calling the `Execute` method.

5.2.1.2.10  MovedCount Property

Used to get the number of records that were read from the source dataset during the batch operation.

Class

`TCRBatchMove`

Syntax

```java
property MovedCount: Integer;
```
Remarks
Use the MovedCount property to get the number of records that were read from the source dataset during the batch operation. This number includes records that caused key or integrity violations or were trimmed.

Class
TCRBatchMove

Syntax

```property
ProblemCount: Integer;
```

Remarks
Use the ProblemCount property to get the number of records that could not be added to the destination dataset because of the field type mismatch.

If AbortOnProblem is True, then ProblemCount will never exceed one, because the operation aborts when the problem occurs.

See Also
- AbortOnProblem

Class
TCRBatchMove

Remarks
Use the RecordCount property to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.

Class
TCRBatchMove
## Syntax

```property```

```RecordCount: Integer default 0;```

## Remarks

Determines the maximum number of records in the source dataset, that will be applied to the destination dataset. If it is set to 0, all records in the source dataset will be applied to the destination dataset, starting from the first record. If RecordCount is greater than 0, up to the RecordCount records are applied to the destination dataset, starting from the current record in the source dataset. If RecordCount exceeds the number of records left in the source dataset, batch operation terminates after reaching last record.

## Class

**TCRBatchMove**

## Syntax

```property```

```Source: TDataSet;```

## Remarks

Specifies the source dataset for the batch operation.

## Methods

Methods of the **TCRBatchMove** class.

For a complete list of the **TCRBatchMove** class members, see the **TCRBatchMove Members** topic.
### Execute Method

Performs the batch operation.

#### Class

**TCRBatchMove**

#### Syntax

```delphi
procedure Execute;
```

#### Remarks

Call the `Execute` method to perform the batch operation.

### Events

Events of the `TCRBatchMove` class.

For a complete list of the `TCRBatchMove` class members, see the `TCRBatchMove Members` topic.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnBatchMoveProgress</td>
<td>Occurs when providing feedback to the user about the batch operation in</td>
</tr>
</tbody>
</table>
5.2.1.4.1 OnBatchMoveProgress Event

Occurs when providing feedback to the user about the batch operation in progress is needed.

Class

TCRBatchMove

Syntax

```pascal
property OnBatchMoveProgress: TCRBatchMoveProgressEvent;
```

Remarks

Write the OnBatchMoveProgress event handler to provide feedback to the user about the batch operation progress.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMoveProgressEvent</td>
<td>This type is used for the TCRBatchMove.OnBatchMoveProgress event.</td>
</tr>
</tbody>
</table>

5.2.2 Types

Types in the CRBatchMove unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMoveProgressEvent</td>
<td>This type is used for the TCRBatchMove.OnBatchMoveProgress event.</td>
</tr>
</tbody>
</table>
5.2.2.1 TCRBatchMoveProgressEvent Procedure Reference

This type is used for the TCRBatchMove.OnBatchMoveProgress event.

Unit
CRBatchMove

Syntax

```
TCRBatchMoveProgressEvent = procedure (Sender: TObject; Percent: integer) of object;
```

Parameters

Sender
An object that raised the event.

Percent
Percentage of the batch operation progress.

5.2.3 Enumerations

Enumerations in the CRBatchMove unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRBatchMode</td>
<td>Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.</td>
</tr>
<tr>
<td>TCRFieldMappingMode</td>
<td>Used to specify the way fields of the destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.</td>
</tr>
</tbody>
</table>
5.2.3.1 TCRBatchMode Enumeration

Used to set the type of the batch operation that will be executed after calling the `TCRBatchMove.Execute` method.

**Unit**

`CRBatchMove`

**Syntax**

```
TCRBatchMode = (bmAppend, bmUpdate, bmAppendUpdate, bmDelete);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmAppend</td>
<td>Appends the records from the source dataset to the destination dataset. The default mode.</td>
</tr>
<tr>
<td>bmAppendUpdate</td>
<td>Replaces records in the destination dataset with the matching records from the source dataset. If there is no matching record in the destination dataset, the record will be appended to it.</td>
</tr>
<tr>
<td>bmDelete</td>
<td>Deletes records from the destination dataset if there are matching records in the source dataset.</td>
</tr>
<tr>
<td>bmUpdate</td>
<td>Replaces records in the destination dataset with the matching records from the source dataset.</td>
</tr>
</tbody>
</table>

5.2.3.2 TCRFieldMappingMode Enumeration

Used to specify the way fields of the destination and source datasets will be mapped to each other if the `TCRBatchMove.Mappings` list is empty.

**Unit**

`CRBatchMove`

**Syntax**

```
TCRFieldMappingMode = (mmFieldIndex, mmFieldName);
```

**Values**
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>mmFieldIndex</td>
<td>Specifies that the fields of the destination dataset will be mapped to the fields of the source dataset by field index.</td>
</tr>
<tr>
<td>mmFieldName</td>
<td>Mapping is performed by field names.</td>
</tr>
</tbody>
</table>

5.3 **CREncryption**

This unit contains base classes for data encryption.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCREncryptor</td>
<td>The class that performs data encryption and decryption in a client application using various encryption algorithms.</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCREncDataHeader</td>
<td>Specifies whether the additional information is stored with the encrypted data.</td>
</tr>
<tr>
<td>TCREncryptionAlgorithm</td>
<td>Specifies the algorithm of data encryption.</td>
</tr>
<tr>
<td>TCRHashAlgorithm</td>
<td>Specifies the algorithm of generating hash data.</td>
</tr>
<tr>
<td>TCRInvalidHashAction</td>
<td>Specifies the action to perform on data fetching when hash data is invalid.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.
5.3.1 Classes

Classes in the CREncryption unit.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCREncryptor</td>
<td>The class that performs data encryption and decryption in a client application using various encryption algorithms.</td>
</tr>
</tbody>
</table>

5.3.1.1 TCREncryptor Class

The class that performs data encryption and decryption in a client application using various encryption algorithms.

For a list of all members of this type, see TCREncryptor members.

### Unit

CREncryption

### Syntax

```delphi
TCREncryptor = class(TComponent);
```

### Members

TCREncryptor class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataHeader</td>
<td>Specifies whether the additional information is stored with the encrypted data.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EncryptionAlgorithm</td>
<td>Specifies the algorithm of data encryption.</td>
</tr>
<tr>
<td>HashAlgorithm</td>
<td>Specifies the algorithm of generating hash data.</td>
</tr>
<tr>
<td>InvalidHashAction</td>
<td>Specifies the action to perform on data fetching when hash data is invalid.</td>
</tr>
<tr>
<td>Password</td>
<td>Used to set a password that is used to generate a key for encryption.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetKey</td>
<td>Sets a key, using which data is encrypted.</td>
</tr>
</tbody>
</table>

Properties of the TCREncryptor class.

For a complete list of the TCREncryptor class members, see the TCREncryptor Members topic.

## Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataHeader</td>
<td>Specifies whether the additional information is stored with the encrypted data.</td>
</tr>
<tr>
<td>EncryptionAlgorithm</td>
<td>Specifies the algorithm of data encryption.</td>
</tr>
<tr>
<td>HashAlgorithm</td>
<td>Specifies the algorithm of generating hash data.</td>
</tr>
<tr>
<td>InvalidHashAction</td>
<td>Specifies the action to perform on data fetching when hash data is invalid.</td>
</tr>
<tr>
<td>Password</td>
<td>Used to set a password that is used to generate a key for encryption.</td>
</tr>
</tbody>
</table>
5.3.1.1.2.1  DataHeader Property

Specifies whether the additional information is stored with the encrypted data.

Class

TCREncryptor

Syntax

```
property DataHeader: TCREncDataHeader default ehTagAndHash;
```

Remarks

Use DataHeader to specify whether the additional information is stored with the encrypted data. Default value is `ehTagAndHash`.

5.3.1.1.2.2  EncryptionAlgorithm Property

Specifies the algorithm of data encryption.

Class

TCREncryptor

Syntax

```
property EncryptionAlgorithm: TCREncryptionAlgorithm default eaBlowfish;
```

Remarks
Use EncryptionAlgorithm to specify the algorithm of data encryption. Default value is \texttt{eaBlowfish}.

5.3.1.1.2.3 HashAlgorithm Property

Specifies the algorithm of generating hash data.

Class

\texttt{TCREncryptor}

Syntax

\begin{verbatim}
property HashAlgorithm: TCRHashAlgorithm default haSHA1;
\end{verbatim}

Remarks

Use HashAlgorithm to specify the algorithm of generating hash data. This property is used only if hash is stored with the encrypted data (the \texttt{DataHeader} property is set to \texttt{ehTagAndHash}). Default value is \texttt{haSHA1}.

5.3.1.1.2.4 InvalidHashAction Property

Specifies the action to perform on data fetching when hash data is invalid.

Class

\texttt{TCREncryptor}

Syntax

\begin{verbatim}
property InvalidHashAction: TCRInvalidHashAction default ihFail;
\end{verbatim}

Remarks

Use InvalidHashAction to specify the action to perform on data fetching when hash data is invalid. This property is used only if hash is stored with the encrypted data (the \texttt{DataHeader} property is set to \texttt{ehTagAndHash}). Default value is \texttt{ihFail}.
If the DataHeader property is set to ehTagAndHash, then on data fetching from a server the hash check is performed for each record. After data decryption its hash is calculated and compared with the hash stored in the field. If these values don’t coincide, it means that the stored data is incorrect, and depending on the value of the InvalidHashAction property one of the following actions is performed:

- **ihFail** - the EInvalidHash exception is raised and further data reading from the server is interrupted.
- **ihSkipData** - the value of the field for this record is set to Null. No exception is raised.
- **ihIgnoreError** - in spite of the fact that the data is not valid, the value is set in the field. No exception is raised.

### 5.3.1.2.5 Password Property

Used to set a password that is used to generate a key for encryption.

**Class**

**TCREncryptor**

**Syntax**

```plaintext
property Password: string stored False;
```

**Remarks**

Use Password to set a password that is used to generate a key for encryption.

**Note:** Calling of the SetKey method clears the Password property.
Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetKey</td>
<td>Sets a key, using which data is encrypted.</td>
</tr>
</tbody>
</table>

See Also
- TCREncryptor Class
- TCREncryptor Class Members

5.3.1.3.1 SetKey Method

Sets a key, using which data is encrypted.

Class
- TCREncryptor

Syntax

```pascal
procedure SetKey(const Key; Count: Integer); overload;
procedure SetKey(const Key: TBytes; Offset: Integer; Count: Integer); overload;
```

Parameters

- **Key**
  - Holds bytes that represent a key.
- **Offset**
  - Offset in bytes to the position, where the key begins.
- **Count**
  - Number of bytes to use from Key.

Remarks

Use SetKey to set a key, using which data is encrypted.

**Note:** Calling of the SetKey method clears the Password property.
5.3.2 Enumerations

Enumerations in the CREncryption unit.

**Enumerations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCREncDataHeader</td>
<td>Specifies whether the additional information is stored with the encrypted data.</td>
</tr>
<tr>
<td>TCREncryptionAlgorithm</td>
<td>Specifies the algorithm of data encryption.</td>
</tr>
<tr>
<td>TCRHashAlgorithm</td>
<td>Specifies the algorithm of generating hash data.</td>
</tr>
<tr>
<td>TCRInvalidHashAction</td>
<td>Specifies the action to perform on data fetching when hash data is invalid.</td>
</tr>
</tbody>
</table>

**5.3.2.1 TCREncDataHeader Enumeration**

Specifies whether the additional information is stored with the encrypted data.

**Unit**

CREncryption

**Syntax**

```plaintext
TCREncDataHeader = (ehTagAndHash, ehTag, ehNone);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ehNone</td>
<td>No additional information is stored.</td>
</tr>
<tr>
<td>ehTag</td>
<td>GUID and the random initialization vector are stored with the encrypted data.</td>
</tr>
<tr>
<td>ehTagAndHash</td>
<td>Hash, GUID, and the random initialization vector are stored with the encrypted data.</td>
</tr>
</tbody>
</table>
5.3.2.2 TCREncryptionAlgorithm Enumeration

Specifies the algorithm of data encryption.

Unit
CREncryption

Syntax
TCREncryptionAlgorithm = (eaTripleDES, eaBlowfish, eaAES128, eaAES192, eaAES256, eaCast128, eaRC4);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>eaAES128</td>
<td>The AES encryption algorithm with key size of 128 bits is used.</td>
</tr>
<tr>
<td>eaAES192</td>
<td>The AES encryption algorithm with key size of 192 bits is used.</td>
</tr>
<tr>
<td>eaAES256</td>
<td>The AES encryption algorithm with key size of 256 bits is used.</td>
</tr>
<tr>
<td>eaBlowfish</td>
<td>The Blowfish encryption algorithm is used.</td>
</tr>
<tr>
<td>eaCast128</td>
<td>The CAST-128 encryption algorithm with key size of 128 bits is used.</td>
</tr>
<tr>
<td>eaRC4</td>
<td>The RC4 encryption algorithm is used.</td>
</tr>
<tr>
<td>eaTripleDES</td>
<td>The Triple DES encryption algorithm is used.</td>
</tr>
</tbody>
</table>

5.3.2.3 TCRHashAlgorithm Enumeration

Specifies the algorithm of generating hash data.

Unit
CREncryption

Syntax
TCRHashAlgorithm = (haSHA1, haMD5);

Values
### 5.3.2.4 TCRInvalidHashAction Enumeration

Specifies the action to perform on data fetching when hash data is invalid.

#### Unit

`CREncryption`

#### Syntax

```pascal
TCRInvalidHashAction = (ihFail, ihSkipData, ihIgnoreError);
```

#### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ihFail</td>
<td>The EInvalidHash exception is raised and further data reading from the server is interrupted.</td>
</tr>
<tr>
<td>ihIgnoreError</td>
<td>In spite of the fact that the data is not valid, the value is set in the field. No exception is raised.</td>
</tr>
<tr>
<td>ihSkipData</td>
<td>The value of the field for this record is set to Null. No exception is raised.</td>
</tr>
</tbody>
</table>

### 5.4 CRGrid

This unit contains the TCRDBGrid component.

#### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRDBGrid</td>
<td>Extends the functionality of the TDBGrid component.</td>
</tr>
</tbody>
</table>
5.4.1 Classes

Classes in the CRGrid unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCRDBGrid</td>
<td>Extends the functionality of the TDBGrid component.</td>
</tr>
</tbody>
</table>

TCRDBGrid Class

Extends the functionality of the TDBGrid component.

For a list of all members of this type, see TCRDBGrid members.

Unit
CRGrid

Syntax

```pascal
TCRDBGrid = class(TCustomDBGrid);
```

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard</td>
</tr>
</tbody>
</table>
### PostgreSQL Data Access Components

TColumn class but is tailored to work with TCRDBGrid only.

| **Filtered** | Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not. |
| **LevelDelimiterChar** | Set LevelDelimiterChar to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns. |
| **OnMemoClick** | Write an OnMemoClick event handler to provide custom processing of Memo fields instead of built-in Memo field editor. |
| **OptionsEx** | OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property. |
| **OptionsMenu** | TCRDBGrid has local menu is situated in the left upper corner. Standard menu has items to show or hide filter bar and search bars. User can change content of this menu using OptionsMenu property. |

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ActivateFilterEdit</strong></td>
<td>Use ActivateFilterEdit at runtime to move input focus to the filter bar. It activates edit control that belongs to the column, specified by Column property.</td>
</tr>
<tr>
<td><strong>ActivateSearchEdit</strong></td>
<td>Use ActivateSearchEdit at</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AdjustColumns</td>
<td>Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.</td>
</tr>
<tr>
<td>ApplyFilter</td>
<td>Call ApplyFilter method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.</td>
</tr>
<tr>
<td>CalcTitleLevel</td>
<td>Call CalcTitleLevel method to set Top and Bottom values of aRect parameter record depending on Level parameter value.</td>
</tr>
<tr>
<td>ClearFilters</td>
<td>Call CalcFilters method to clear all filter statements for grid columns.</td>
</tr>
<tr>
<td>ClearSorting</td>
<td>Call ClearSorting method to discard sorting previously applied to the grid data.</td>
</tr>
<tr>
<td>DataChanged</td>
<td>Call DataChanged method to update displayed data and status information for the grid.</td>
</tr>
<tr>
<td>GetGridSize</td>
<td>Call GetGridSize method to obtain width of a client area for the grid component.</td>
</tr>
<tr>
<td>GetTitleLevel</td>
<td>GetTitleLevel method returns TRect structure filled with top and bottom coordinates for the specified title level.</td>
</tr>
</tbody>
</table>

**Events**
### Properties

Properties of the **TCRDBGrid** class.

For a complete list of the **TCRDBGrid** class members, see the [TCRDBGrid Members](#) topic.

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OptionsMenu</strong></td>
<td>TCRDBGrid has local menu is situated in the left upper corner. Standard menu has items to show or hide filter bar and search bars. User can change content of this menu using OptionsMenu property.</td>
</tr>
</tbody>
</table>

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Columns</strong></td>
<td>To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard TColumn class but is tailored to work with TCRDBGrid only.</td>
</tr>
<tr>
<td><strong>Filtered</strong></td>
<td>Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.</td>
</tr>
</tbody>
</table>
### LevelDelimiterChar

Set `LevelDelimiterChar` to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.

### OnMemoClick

Write an `OnMemoClick` event handler to provide custom processing of Memo fields instead of built-in Memo field editor.

### OptionsEx

OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property.

### See Also

- [TCRDBGrid Class](#)
- [TCRDBGrid Class Members](#)

---

5.4.1.1.2.1 Columns Property

To provide extended functionality TCRDBGrid component is shipped with a TCRColumn class, which is derived from the standard TColumn class but is tailored to work with TCRDBGrid only.

### Class

**TCRDBGrid**

### Syntax

```{}
property Columns: TCRDBGridColumnns stored False;
```

### Remarks

When browsing TDBGridColumnns remember that its Items property actually returns TCRColumn objects when indexed.
5.4.1.2.2 Filtered Property

Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.

Class

TCRDBGrid

Syntax

```
property Filtered: boolean default True;
```

Remarks

Conditional statements of all columns are combined to form a single filter.

5.4.1.2.3 LevelDelimiterChar Property

Set LevelDelimiterChar to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.

Class

TCRDBGrid

Syntax

```
property LevelDelimiterChar: char default '\';
```

5.4.1.2.4 OnMemoClick Property

Write an OnMemoClick event handler to provide custom processing of Memo fields instead of built-in Memo field editor.

Class
TCRDBGrid

Syntax

```pascal
property OnMemoClick: TOnMemoClick;
```

OptionsEx Property

OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property.

Class

TCRDBGrid

Syntax

```pascal
property OptionsEx: TCRDBGridOptionsEx default [dgeEnableSort, dgeLocalFilter, dgeLocalSorting, dgeRecordCount];
```

Remarks

Following values are supported for the OptionsEx set:

- **dgeEnableSort** - enables sorting of records by the specified column clicking on the column title.

- **dgeFilterBar** - shows filter bar below grid caption where filter expressions for every column are entered manually.

- **dgeLocalFilter** - records are filtered using associated dataset component facilities. Otherwise database server processes all filter operations.

- **dgeLocalSorting** - records are sorted using associated dataset component facilities. Otherwise database server processes all sorting operations.

- **dgeRecordCount** - displays status bar with current record number in respect to the total number of records. This option is mutually exclusive with dgeSummary option.

- **dgeSearchBar** - shows search bar below grid caption where incremental search expression is entered for every column.

- **dgeStretch** - makes all columns adjust their widths so that they fit onto entire grid area.
`dgeSummary` - displays status bar with summary information for every column. See TCRColumn.SummaryMode property on description of available options for individual columns. `dgeSummary` is mutually exclusive with `dgeRecordCount` option.

Class

**TCRDBGrid**

Syntax

```property
OptionsMenu: TPopupMenu;
```

Methods of the **TCRDBGrid** class.

For a complete list of the **TCRDBGrid** class members, see the [TCRDBGrid Members](#) topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivateFilterEdit</td>
<td>Use <code>ActivateFilterEdit</code> at run-time to move input focus to the filter bar. It activates edit control that belongs to the column, specified by <code>Column</code> property.</td>
</tr>
<tr>
<td>ActivateSearchEdit</td>
<td>Use <code>ActivateSearchEdit</code> at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AdjustColumns</td>
<td>Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.</td>
</tr>
<tr>
<td>ApplyFilter</td>
<td>Call ApplyFilter method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.</td>
</tr>
<tr>
<td>CalcTitleLevel</td>
<td>Call CalcTitleLevel method to set Top and Bottom values of aRect parameter record depending on Level parameter value.</td>
</tr>
<tr>
<td>ClearFilters</td>
<td>Call CalcFilters method to clear all filter statements for grid columns.</td>
</tr>
<tr>
<td>ClearSorting</td>
<td>Call ClearSorting method to discard sorting previously applied to the grid data.</td>
</tr>
<tr>
<td>DataChanged</td>
<td>Call DataChanged method to update displayed data and status information for the grid.</td>
</tr>
<tr>
<td>GetGridSize</td>
<td>Call GetGridSize method to obtain width of a client area for the grid component.</td>
</tr>
<tr>
<td>GetTitleLevel</td>
<td>GetTitleLevel method returns TRect structure filled with top and bottom coordinates for the specified title level.</td>
</tr>
</tbody>
</table>

See Also
- TCRDBGrid Class
- TCRDBGrid Class Members
5.4.1.1.3.1  ActivateFilterEdit Method

Use ActivateFilterEdit at run-time to move input focus to the filter bar. It activates edit control that belongs to the column, specified by Column property.

Class

TCRDBGrid

Syntax

procedure ActivateFilterEdit(Column: TColumn);

Parameters

Column

© 1997-2021
Devart. All Rights Reserved.

5.4.1.1.3.2  ActivateSearchEdit Method

Use ActivateSearchEdit at run-time to move input focus to the search bar. It activates edit control that belongs to the column, specified by Column property.

Class

TCRDBGrid

Syntax

procedure ActivateSearchEdit(Column: TColumn);

Parameters

Column

© 1997-2021
Devart. All Rights Reserved.

5.4.1.1.3.3  AdjustColumns Method

Call AdjustColumns method at run-time to stretch all columns in a grid so that they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.
Class

**TCRDBGrid**

**Syntax**

```plaintext
procedure AdjustColumns;
```

**Remarks**

Visual feedback is immediate if `OptionsEx` property includes `dgeLocalFilter` option.

**Note:** Since filtering is performed at session level then other data-aware controls may also be updated.

---

© 1997-2021 Devart. All Rights Reserved.  
Request Support  DAC Forum  Provide Feedback

**5.4.1.1.3.4 ApplyFilter Method**

Call `ApplyFilter` method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.

Class

**TCRDBGrid**

**Syntax**

```plaintext
procedure ApplyFilter;
```

**Remarks**

Visual feedback is immediate if `OptionsEx` property includes `dgeLocalFilter` option.

**Note:** Since filtering is performed at session level then other data-aware controls may also be updated.

---

© 1997-2021 Devart. All Rights Reserved.  
Request Support  DAC Forum  Provide Feedback

**5.4.1.1.3.5 CalcTitleLevel Method**

Call `CalcTitleLevel` method to set Top and Bottom values of `aRect` parameter record depending on Level parameter value.
Class
TCRDBGrid

Syntax

```procedure CalcTitleLevel(Level: integer; var aRect: TRect);```

Parameters

- `Level`
- `aRect`

5.4.1.3.6 ClearFilters Method

Call CalcFilters method to clear all filter statements for grid columns.

Class
TCRDBGrid

Syntax

```procedure ClearFilters;```

5.4.1.3.7 ClearSorting Method

Call ClearSorting method to discard sorting previously applied to the grid data.

Class
TCRDBGrid

Syntax

```procedure ClearSorting;```
5.4.1.3.8 DataChanged Method

Call DataChanged method to update displayed data and status information for the grid.

Class

TCRDBGrid

Syntax

```
procedure DataChanged;
```

5.4.1.3.9 GetGridSize Method

Call GetGridSize method to obtain width of a client area for the grid component.

Class

TCRDBGrid

Syntax

```
function GetGridSize: integer;
```

Remarks

Client area accommodates cell values only without column grid lines.

5.4.1.3.10 GetTitleLevel Method

GetTitleLevel method returns TRect structure filled with top and bottom coordinates for the specified title level.

Class

TCRDBGrid

Syntax

```
function GetTitleLevel(Level: integer): TRect;
```
5.4.1.4 Events

Events of the TCRDBGrid class.

For a complete list of the TCRDBGrid class members, see the TCRDBGrid Members topic.

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnGetCellParams</td>
<td>Write an OnGetCellParams event handler to provide custom background colors while the grid redraws individual cells.</td>
</tr>
</tbody>
</table>

See Also

- TCRDBGrid Class
- TCRDBGrid Class Members

Class

TCRDBGrid

Syntax

```property
OnGetCellParams: TGetCellParamsEvent;
```
5.5 CRVio

This unit contains classes for HTTP connections.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THttpOptions</td>
<td>This class is used to establish an HTTP connection.</td>
</tr>
<tr>
<td>TProxyOptions</td>
<td>This class is used to establish an HTTP connection through a proxy server.</td>
</tr>
</tbody>
</table>

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIPVersion</td>
<td>Specifies Internet Protocol version.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.5.1 Classes

Classes in the CRVio unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THttpOptions</td>
<td>This class is used to establish an HTTP connection.</td>
</tr>
<tr>
<td>TProxyOptions</td>
<td>This class is used to establish an HTTP connection through a proxy server.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
5.5.1.1 **THttpOptions Class**

This class is used to establish an HTTP connection.

For a list of all members of this type, see **THttpOptions** members.

**Unit**

*CRVio*

**Syntax**

```cpp
THttpOptions = class(TPersistent);
```

**Remarks**

The THttpOptions class is used to establish an HTTP connection.

For more information about HTTP tunneling, see [Network Tunneling](#).

**See Also**

- [Network Tunneling](#)

© 1997-2021 Devart. All Rights Reserved.

### 5.5.1.1.1 Members

**THttpOptions** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enabled</strong></td>
<td>Enables an HTTP connection.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Holds the password for HTTP authorization.</td>
</tr>
<tr>
<td><strong>ProxyOptions</strong></td>
<td>Holds a TProxyOptions object that contains settings for a proxy connection.</td>
</tr>
<tr>
<td><strong>TrustServerCertificate</strong></td>
<td>Verifies the server certificate during an SSL handshake.</td>
</tr>
<tr>
<td><strong>Url</strong></td>
<td>Holds the URL of the PHP script for HTTP tunneling.</td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Holds the username for</td>
</tr>
</tbody>
</table>
### Properties of the `THttpOptions` class.

For a complete list of the `THttpOptions` class members, see the [THttpOptions Members](#) topic.

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enabled</strong></td>
<td>Enables an HTTP connection.</td>
</tr>
<tr>
<td><strong>ProxyOptions</strong></td>
<td>Holds a <code>TProxyOptions</code> object that contains settings for a proxy connection.</td>
</tr>
</tbody>
</table>

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Password</strong></td>
<td>Holds the password for HTTP authorization.</td>
</tr>
<tr>
<td><strong>TrustServerCertificate</strong></td>
<td>Verifies the server certificate during an SSL handshake.</td>
</tr>
<tr>
<td><strong>Url</strong></td>
<td>Holds the URL of the PHP script for HTTP tunneling.</td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Holds the username for HTTP authorization.</td>
</tr>
</tbody>
</table>

### See Also

- [THttpOptions Class](#)
- [THttpOptions Class Members](#)
5.5.1.1.2.1 Enabled Property

Enables an HTTP connection.

Class

THttpOptions

Syntax

property Enabled: boolean default False;

Remarks

The Enabled property specifies that a connection is established through HTTP.

5.5.1.1.2.2 Password Property

Holds the password for HTTP authorization.

Class

THttpOptions

Syntax

property Password: string;

Remarks

The Password property holds the password for the password-protected directory that contains the HTTP tunneling script.

5.5.1.1.2.3 ProxyOptions Property

Holds a TProxyOptions object that contains settings for a proxy connection.

Class

THttpOptions
5.5.1.2.4 TrustServerCertificate Property

Verifies the server certificate during an SSL handshake.

Class

THttpOptions

Syntax

| property | TrustServerCertificate: boolean default False |

Remarks

The TrustServerCertificate property specifies whether to verify the server certificate during an SSL handshake. When True, the PgDac bypasses walking the certificate chain to verify the certificate. The default value is False.

© 1997-2021
Devart. All Rights Reserved.

5.5.1.2.5 Url Property

Holds the URL of the PHP script for HTTP tunneling.

Class

THttpOptions

© 2021 Devart
5.5.1.2.6 Username Property

Holds the username for HTTP authorization.

Class

THttpOptions

Syntax

```property Username: string;```

Remarks

The Username property holds the username for the password-protected directory that contains the HTTP tunneling script.

5.5.1.2 TProxyOptions Class

This class is used to establish an HTTP connection through a proxy server.

For a list of all members of this type, see TProxyOptions members.

Unit

CRVio

Syntax

```TProxyOptions = class(TPersistent);```
### Remarks

The TProxyOptions class is used to establish an HTTP connection through a proxy server.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname</td>
<td>Holds the hostname or IP address of the proxy server.</td>
</tr>
<tr>
<td>Password</td>
<td>Holds the proxy password.</td>
</tr>
<tr>
<td>Port</td>
<td>Holds the port number of the proxy server.</td>
</tr>
<tr>
<td>Username</td>
<td>Holds the proxy username.</td>
</tr>
</tbody>
</table>

## Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname</td>
<td>Holds the hostname or IP address of the proxy server.</td>
</tr>
<tr>
<td>Password</td>
<td>Holds the proxy password.</td>
</tr>
<tr>
<td>Port</td>
<td>Holds the port number of the proxy server.</td>
</tr>
<tr>
<td>Username</td>
<td>Holds the proxy username.</td>
</tr>
</tbody>
</table>
See Also

- **TProxyOptions Class**
- **TProxyOptions Class Members**

### 5.5.1.2.2.1 Hostname Property

Holds the hostname or IP address of the proxy server.

**Class**

**TProxyOptions**

**Syntax**

```
property Hostname: string;
```

**Remarks**

The Hostname property holds the hostname or IP address of the proxy server.

### 5.5.1.2.2.2 Password Property

Holds the proxy password.

**Class**

**TProxyOptions**

**Syntax**

```
property Password: string;
```

**Remarks**

The Password property holds the proxy password.
5.5.1.2.2.3 Port Property

Holds the port number of the proxy server.

Class
TProxyOptions

Syntax

| property | Port: integer default 0; |

Remarks
Use the Port property to specify the port number of the proxy server.

5.5.1.2.2.4 Username Property

Holds the proxy username.

Class
TProxyOptions

Syntax

| property | Username: string |

Remarks
The Username property holds the proxy username.

5.5.2 Enumerations

Enumerations in the CRVio unit.

Enumerations

| Name | Description |
## 5.5.2.1 TIPVersion Enumeration

Specifies Internet Protocol version.

### Unit

**CRVio**

### Syntax

```plaintext
TIPVersion = (ivIPv4, ivIPv6, ivIPBoth);
```

### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivIPBoth</td>
<td>Specifies that either IPv6 or IPv4 Internet Protocol version is used</td>
</tr>
<tr>
<td>ivIPv4</td>
<td>Specifies that the IPv4 Internet Protocol version is used</td>
</tr>
<tr>
<td>ivIPv6</td>
<td>Specifies that the IPv6 Internet Protocol version is used</td>
</tr>
</tbody>
</table>

### Remarks

**Note:** When the TIPVersion property is set to *ivIPBoth*, a connection attempt is made via IPv6 if it is enabled in the operating system settings. If the connection attempt fails, a new connection attempt is made via IPv4.

### See Also

- `TPgConnectionOptions.IPVersion`

## 5.6 DAAlerter

This unit contains the base class for the TPgAlerter component.

### Classes
### 5.6.1 Classes

Classes in the `DAAlerter` unit.

#### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TDAAlerter</code></td>
<td>A base class that defines functionality for database event notification.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

### 5.6.1.1 TDAAlerter Class

A base class that defines functionality for database event notification.

For a list of all members of this type, see `TDAAlerter` members.

**Unit**

`DAAlerter`

**Syntax**

```delphi
TDAAlerter = class(TComponent);
```

**Remarks**
TDAAlerter is a base class that defines functionality for descendant classes support database event notification. Applications never use TDAAlerter objects directly. Instead they use descendants of TDAAlerter.

The TDAAlerter component allows you to register interest in and handle events posted by a database server. Use TDAAlerter to handle events for responding to actions and database changes made by other applications. To get events, an application must register required events. To do this, set the Events property to the required events and call the Start method. When one of the registered events occurs OnEvent handler is called.

### Members

#### TDAAlerter class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if TDAAlerter waits for messages.</td>
</tr>
<tr>
<td>AutoRegister</td>
<td>Used to automatically register events whenever connection opens.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify the connection for TDAAlerter.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendEvent</td>
<td>Sends an event with Name and content Message.</td>
</tr>
<tr>
<td>Start</td>
<td>Starts waiting process.</td>
</tr>
<tr>
<td>Stop</td>
<td>Stops waiting process.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnError</td>
<td>Occurs if an exception</td>
</tr>
</tbody>
</table>
Properties

Properties of the TDAAlerter class.

For a complete list of the TDAAlerter class members, see the TDAAlerter Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if TDAAlerter waits for messages.</td>
</tr>
<tr>
<td>AutoRegister</td>
<td>Used to automatically register events whenever connection opens.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify the connection for TDAAlerter.</td>
</tr>
</tbody>
</table>

See Also
- TDAAlerter Class
- TDAAlerter Class Members

Remarks

Used to determine if TDAAlerter waits for messages.
Check the Active property to know whether TDAlerter waits for messages or not. Set it to True to register events.

See Also
- Start
- Stop

AutoRegister Property

Used to automatically register events whenever connection opens.

Class
TDAAlerter

Syntax
```
property AutoRegister: boolean default False;
```

Remarks
Set the AutoRegister property to True to automatically register events whenever connection opens.

Connection Property

Used to specify the connection for TDAAlerter.

Class
TDAAlerter

Syntax
```
property Connection: TCustomDAConnection;
```

Remarks
Use the Connection property to specify the connection for TDAAlerter.
5.6.1.1.3 Methods

Methods of the **TDAAlerter** class.

For a complete list of the **TDAAlerter** class members, see the [TDAAlerter Members](#) topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SendEvent</strong></td>
<td>Sends an event with Name and content Message.</td>
</tr>
<tr>
<td><strong>Start</strong></td>
<td>Starts waiting process.</td>
</tr>
<tr>
<td><strong>Stop</strong></td>
<td>Stops waiting process.</td>
</tr>
</tbody>
</table>

**See Also**
- [TDAAlerter Class](#)
- [TDAAlerter Class Members](#)

**5.6.1.1.3.1 SendEvent Method**

Sends an event with Name and content Message.

**Class**

**TDAAlerter**

**Syntax**

```plaintext
procedure SendEvent(const EventName: string; const Message: string);
```

**Parameters**

*EventName*
- Holds the event name.

*Message*
- Holds the content Message of the event.
5.6.1.1.3.2 Start Method

Starts waiting process.

Class

TDAAlerter

Syntax

```pascal
procedure Start;
```

Remarks

Call the Start method to run waiting process. After starting TDAAlerter waits for messages with names defined by the Events property.

See Also

- Stop
- Active

5.6.1.1.3.3 Stop Method

Stops waiting process.

Class

TDAAlerter

Syntax

```pascal
procedure Stop;
```

Remarks
Call Stop method to end waiting process.

See Also
- Start

Events of the **TDAAlerter** class.

For a complete list of the **TDAAlerter** class members, see the [TDAAlerter Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnError</strong></td>
<td>Occurs if an exception occurs in waiting process</td>
</tr>
</tbody>
</table>

See Also
- [TDAAlerter Class](#)
- [TDAAlerter Class Members](#)

### 5.6.1.1.4.1 OnError Event

Occurs if an exception occurs in waiting process

**Class**

**TDAAlerter**

**Syntax**

```object
property OnError: TAlertErrorEvent;
```

**Remarks**

The OnError event occurs if an exception occurs in waiting process. Alerter stops in this case. The exception can be accessed using the E parameter.
5.6.2 Types

Types in the DAAlerter unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAlerterErrorEvent</td>
<td>This type is used for the TDAAlerter.OnError event.</td>
</tr>
</tbody>
</table>

5.6.2.1 TAlerterErrorEvent Procedure Reference

This type is used for the TDAAlerter.OnError event.

Unit

DAAlerter

Syntax

```
TAlerterErrorEvent = procedure (Sender: TDAAlerter; E: Exception) of object;
```

Parameters

Sender
An object that raised the event.

E
Exception object.

5.7 DADump

This unit contains the base class for the TPgDump component.

Classes
### Classes

Classes in the `DADump` unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDADump</td>
<td>A base class that defines functionality for descendant classes that dump database objects to a script.</td>
</tr>
<tr>
<td>TDADumpOptions</td>
<td>This class allows setting up the behaviour of the TDADump class.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDABackupProgressEvent</td>
<td>This type is used for the <code>TDADump.OnBackupProgress</code> event.</td>
</tr>
<tr>
<td>TDARestoreProgressEvent</td>
<td>This type is used for the <code>TDADump.OnRestoreProgress</code> event.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  
[Request Support]  [DAC Forum]  [Provide Feedback]
5.7.1.1 TDADump Class

A base class that defines functionality for descendant classes that dump database objects to a script.

For a list of all members of this type, see TDADump members.

Unit

DADump

Syntax

TDADump = class(TComponent);

Remarks

TDADump is a base class that defines functionality for descendant classes that dump database objects to a script. Applications never use TDADump objects directly. Instead they use descendants of TDADump.

Use TDADump descendants to dump database objects, such as tables, stored procedures, and functions for backup or for transferring the data to another SQL server. The dump contains SQL statements to create the table or other database objects and/or populate the table.

© 1997-2021 Devart. All Rights Reserved.

5.7.1.1.1 Members

TDADump class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Used to specify a connection object that will be used to connect to a data store.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
</tbody>
</table>
### Options
- **Options**
  - Used to specify the behaviour of a TDADump component.

### SQL
- **SQL**
  - Used to set or get the dump script.

### TableNames
- **TableNames**
  - Used to set the names of the tables to dump.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup</td>
<td>Dumps database objects to the <code>TDADump.SQL</code> property.</td>
</tr>
<tr>
<td>BackupQuery</td>
<td>Dumps the results of a particular query.</td>
</tr>
<tr>
<td>BackupToFile</td>
<td>Dumps database objects to the specified file.</td>
</tr>
<tr>
<td>BackupToStream</td>
<td>Dumps database objects to the stream.</td>
</tr>
<tr>
<td>Restore</td>
<td>Executes a script contained in the SQL property.</td>
</tr>
<tr>
<td>RestoreFromFile</td>
<td>Executes a script from a file.</td>
</tr>
<tr>
<td>RestoreFromStream</td>
<td>Executes a script received from the stream.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnError</td>
<td>Occurs when PostgreSQL raises some error on <code>TDADump.Restore</code>.</td>
</tr>
<tr>
<td>OnRestoreProgress</td>
<td>Occurs to indicate the</td>
</tr>
</tbody>
</table>
5.7.1.1.2 Properties

Properties of the **TDADump** class.

For a complete list of the **TDADump** class members, see the [TDADump Members](#) topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify a connection object that will be used to connect to a data store.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to specify the behaviour of a TDADump component.</td>
</tr>
</tbody>
</table>

**Published**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Debug</strong></td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to set or get the dump script.</td>
</tr>
<tr>
<td><strong>TableNames</strong></td>
<td>Used to set the names of the tables to dump.</td>
</tr>
</tbody>
</table>

See Also

- [TDADump Class](#)
- [TDADump Class Members](#)
5.7.1.1.2.1 Connection Property

Used to specify a connection object that will be used to connect to a data store.

Class

TDADump

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

See Also

• TCustomDAConnection

5.7.1.1.2.2 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

TDADump

Syntax

```
property Debug: boolean default False;
```
Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

**Note:** If TPgSQLMonitor is used in the project and the TPgSQLMonitor.Active property is set to False, the debug window is not displayed.

See Also

- TCustomDADataset.Debug
- TCustomDASQL.Debug

© 1997-2021 Devart. All Rights Reserved.

5.7.1.2.3 Options Property

Used to specify the behaviour of a TDADump component.

Class

TDADump

Syntax

```property Options: TDADumpOptions;```

Remarks

Use the Options property to specify the behaviour of a TDADump component.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddDrop</td>
<td>Used to add drop statements to a script before creating statements.</td>
</tr>
<tr>
<td>CompleteInsert</td>
<td>Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.</td>
</tr>
<tr>
<td>GenerateHeader</td>
<td>Used to add a comment header to a script.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used for TDADump to quote all database</td>
</tr>
</tbody>
</table>
object names in generated SQL statements.

5.7.1.1.2.4 SQL Property

Used to set or get the dump script.

Class

TDADump

Syntax

| property | SQL: TStrings; |

Remarks

Use the SQL property to get or set the dump script. The SQL property stores script that is executed by the Restore method. This property will store the result of Backup and BackupQuery. At design time the SQL property can be edited by invoking the String List editor in Object Inspector.

See Also

- Restore
- Backup
- BackupQuery

5.7.1.1.2.5 TableNames Property

Used to set the names of the tables to dump.

Class

TDADump

Syntax

| property | TableNames: string; |
Remarks

Use the TableNames property to set the names of the tables to dump. Table names must be separated with semicolons. If the property is empty, the Backup method will dump all available tables.

See Also

- Backup

5.7.1.1.3 Methods

Methods of the TDADump class.

For a complete list of the TDADump class members, see the TDADump Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup</td>
<td>Dumps database objects to the TDADump.SQL property.</td>
</tr>
<tr>
<td>BackupQuery</td>
<td>Dumps the results of a particular query.</td>
</tr>
<tr>
<td>BackupToFile</td>
<td>Dumps database objects to the specified file.</td>
</tr>
<tr>
<td>BackupToStream</td>
<td>Dumps database objects to the stream.</td>
</tr>
<tr>
<td>Restore</td>
<td>Executes a script contained in the SQL property.</td>
</tr>
<tr>
<td>RestoreFromFile</td>
<td>Executes a script from a file.</td>
</tr>
<tr>
<td>RestoreFromStream</td>
<td>Executes a script received from the stream.</td>
</tr>
</tbody>
</table>

See Also

- TDADump Class
- TDADump Class Members
5.7.1.1.3.1 Backup Method

Dumps database objects to the SQL property.

Class
TDADump

Syntax

```plaintext
procedure Backup;
```

Remarks
Call the Backup method to dump database objects. The result script will be stored in the SQL property.

See Also
- SQL
- Restore
- BackupToFile
- BackupToStream
- BackupQuery

5.7.1.3.2 BackupQuery Method

Dumps the results of a particular query.

Class
TDADump

Syntax

```plaintext
procedure BackupQuery(const Query: string);
```

Parameters
Query
Holds a query used for data selection.
Remarks

Call the BackupQuery method to dump the results of a particular query. Query must be a valid select statement. If this query selects data from several tables, only data of the first table in the from list will be dumped.

See Also

- Restore
- Backup
- BackupToFile
- BackupToStream

© 1997-2021 Devart. All Rights Reserved.

5.7.1.1.3.3 BackupToFile Method

Dumps database objects to the specified file.

Class

TDADump

Syntax

```procedure BackupToFile(const FileName: string; const Query: string = '');```

Parameters

- **FileName**
  
  Holds the file name to dump database objects to.

- **Query**
  
  Your query to receive the data for dumping.

Remarks

Call the BackupToFile method to dump database objects to the specified file.

See Also

- RestoreFromStream
- Backup
- BackupToStream
5.7.1.3.4  BackupToStream Method

Dumps database objects to the stream.

Class

TDADump

Syntax

```
procedure BackupToStream(Stream: TStream; const Query: string = '');
```

Parameters

- **Stream**
  - Holds the stream to dump database objects to.
- **Query**
  - Your query to receive the data for dumping.

Remarks

Call the BackupToStream method to dump database objects to the stream.

See Also

- RestoreFromStream
- Backup
- BackupToFile

5.7.1.3.5  Restore Method

Executes a script contained in the SQL property.

Class

TDADump

Syntax
procedure Restore;

Remarks

Call the Restore method to execute a script contained in the SQL property.

See Also

- RestoreFromFile
- RestoreFromStream
- Backup
- SQL

5.7.1.1.3.6 RestoreFromFile Method

Executes a script from a file.

Class

TDADump

Syntax

procedure RestoreFromFile(const FileName: string);

Parameters

FileName

Holds the file name to execute a script from.

Remarks

Call the RestoreFromFile method to execute a script from the specified file.

See Also

- Restore
- RestoreFromStream
- BackupToFile
5.7.1.1.3.7  RestoreFromStream Method

Executes a script received from the stream.

Class
TDADump

Syntax

```pascal
procedure RestoreFromStream(Stream: TStream);
```

Parameters

`Stream`
Holds a stream to receive a script to be executed.

Remarks

Call the RestoreFromStream method to execute a script received from the stream.

See Also

- Restore
- RestoreFromFile
- BackupToStream

© 1997-2021
Devart. All Rights Reserved.

5.7.1.1.4  Events

Events of the `TDADump` class.

For a complete list of the `TDADump` class members, see the `TDADump Members` topic.

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
### 5.1.1.4.1 OnBackupProgress Event


**Class**

`TDADump`

**Syntax**

```property
property OnBackupProgress: TDABackupProgressEvent;
```

**Remarks**

The `OnBackupProgress` event occurs several times during the dumping process of the `Backup`, `M:Devart.Dac.TDADump.BackupToFile(System.String)`, or `M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream)` method execution and indicates its progress. `ObjectName` parameter indicates the name of the currently dumping database object. `ObjectNum` shows the number of the current database object in the backup queue starting from zero. `ObjectCount` shows the quantity of database objects to dump. `Percent` parameter shows the current percentage of the current table data dumped, not the.
current percentage of the entire dump process.

See Also
• Backup
• BackupToFile
• BackupToStream

5.7.1.1.4.2 OnError Event

Occurs when PostgreSQL raises some error on Restore.

Class
TDADump

Syntax

property OnError: TOnErrorEvent;

Remarks
The OnError event occurs when PostgreSQL raises some error on Restore.

Action indicates the action to take when the OnError handler exits. On entry into the handler, Action is always set to eaException.

Note: You should add the DAScript module to the 'uses' list to use the OnError event handler.

5.7.1.1.4.3 OnRestoreProgress Event

Occurs to indicate the Restore, RestoreFromFile, or RestoreFromStream method execution progress.

Class
TDADump

Syntax
**property** OnRestoreProgress: TDARestoreProgressEvent;

Remarks

The OnRestoreProgress event occurs several times during the dumping process of the Restore, RestoreFromFile, or RestoreFromStream method execution and indicates its progress. The Percent parameter of the OnRestoreProgress event handler indicates the percentage of the whole restore script execution.

See Also

- Restore
- RestoreFromFile
- RestoreFromStream

© 1997-2021 Devart. All Rights Reserved.

5.7.1.2 TDADumpOptions Class

This class allows setting up the behaviour of the TDADump class.

For a list of all members of this type, see TDADumpOptions members.

Unit

DADump

Syntax

TDADumpOptions = class(TPersistent);

© 1997-2021 Devart. All Rights Reserved.

5.7.1.2.1 Members

**TDADumpOptions** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddDrop</td>
<td>Used to add drop statements to a script before creating statements.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AddDrop</td>
<td>Used to add drop statements to a script before creating statements.</td>
</tr>
<tr>
<td>CompleteInsert</td>
<td>Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.</td>
</tr>
<tr>
<td>GenerateHeader</td>
<td>Used to add a comment header to a script.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used for TDADump to quote all database object names in generated SQL statements.</td>
</tr>
</tbody>
</table>

Properties of the `TDADumpOptions` class.

For a complete list of the `TDADumpOptions` class members, see the `TDADumpOptions Members` topic.

Published

See Also
- `TDADumpOptions Class`
- `TDADumpOptions Class Members`
5.7.1.2.2.1 AddDrop Property

Used to add drop statements to a script before creating statements.

Class

TDADumpOptions

Syntax

```text
property AddDrop: boolean default True;
```

Remarks

Use the AddDrop property to add drop statements to a script before creating statements.

© 1997-2021
Devart. All Rights Reserved.

5.7.1.2.2.2 CompleteInsert Property

Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.

Class

TDADumpOptions

Syntax

```text
property CompleteInsert: boolean default False;
```

Remarks

If the CompleteInsert property is set to True, SQL query will include the field names, for example:

```
INSERT INTO dept(deptno, dname, loc) VALUES ('10', 'ACCOUNTING', 'NEW YORK');
```

If False, it won't include the field names, for example:

```
INSERT INTO dept VALUES ('10', 'ACCOUNTING', 'NEW YORK');
```
5.7.1.2.2.3 GenerateHeader Property

Used to add a comment header to a script.

Class
TDADumpOptions

Syntax

```property GenerateHeader: boolean default True;```

Remarks
Use the GenerateHeader property to add a comment header to a script. It contains script generation date, DAC version, and some other information.

© 1997-2021 Devart. All Rights Reserved.

5.7.1.2.2.4 QuoteNames Property

Used for TDADump to quote all database object names in generated SQL statements.

Class
TDADumpOptions

Syntax

```property QuoteNames: boolean default False;```

Remarks
If the QuoteNames property is True, TDADump quotes all database object names in generated SQL statements.

© 1997-2021 Devart. All Rights Reserved.

5.7.2 Types

Types in the DADump unit.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDABackupProgressEvent</td>
<td>This type is used for the TDADump.OnBackupProgress event.</td>
</tr>
<tr>
<td>TDARestoreProgressEvent</td>
<td>This type is used for the TDADump.OnRestoreProgress event.</td>
</tr>
</tbody>
</table>

5.7.2.1 TDABackupProgressEvent Procedure Reference

This type is used for the TDADump.OnBackupProgress event.

Unit

DADump

Syntax

```plaintext
TDABackupProgressEvent = procedure (Sender: TObject; ObjectName: string; ObjectNum: integer; ObjectCount: integer; Percent: integer) of object;
```

Parameters

- **Sender**
  - An object that raised the event.
- **ObjectName**
  - The name of the currently dumping database object.
- **ObjectNum**
  - The number of the current database object in the backup queue starting from zero.
- **ObjectCount**
  - The quantity of database objects to dump.
- **Percent**
  - The current percentage of the current table data dumped.
5.7.2.2  TDARestoreProgressEvent Procedure Reference

This type is used for the TDADump.OnRestoreProgress event.

Unit

DADump

Syntax

TDARestoreProgressEvent = procedure (Sender: TObject; Percent: integer) of object;

Parameters

Sender
An object that raised the event.

Percent
The percentage of the whole restore script execution.

5.8  DALoader

This unit contains the base class for the TPgLoader component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAColumn</td>
<td>Represents the attributes for column loading.</td>
</tr>
<tr>
<td>TDAColumns</td>
<td>Holds a collection of TDAColumn objects.</td>
</tr>
<tr>
<td>TDALoader</td>
<td>This class allows loading external data into database.</td>
</tr>
<tr>
<td>TDALoaderOptions</td>
<td>Allows loading external data into database.</td>
</tr>
</tbody>
</table>

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAPutDataEvent</td>
<td>This type is used for the TDALoader.OnPutData event.</td>
</tr>
</tbody>
</table>
5.8.1 Classes

Classes in the DALoader unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAColumn</td>
<td>Represents the attributes for column loading.</td>
</tr>
<tr>
<td>TDAColumns</td>
<td>Holds a collection of TDAColumn objects.</td>
</tr>
<tr>
<td>TDALoader</td>
<td>This class allows loading external data into database.</td>
</tr>
<tr>
<td>TDALoaderOptions</td>
<td>Allows loading external data into database.</td>
</tr>
</tbody>
</table>

5.8.1.1 TDAColumn Class

Represents the attributes for column loading.

For a list of all members of this type, see TDAColumn members.

Unit
DALoader

Syntax

```
TDAColumn = class(TCollectionItem);
```

Remarks
Each `TDALoader` uses `TDAColumns` to maintain a collection of `TDAColumn` objects. `TDAColumn` object represents the attributes for column loading. Every `TDAColumn` object corresponds to one of the table fields with the same name as its `TDAColumn.Name` property.

To create columns at design-time use the column editor of the `TDALoader` component.

**See Also**

- `TDALoader`
- `TDAColumns`

### 5.8.1.1.1 Members

**TDAColumn** class overview.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FieldType</strong></td>
<td>Used to specify the types of values that will be loaded.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Used to specify the field name of loading table.</td>
</tr>
</tbody>
</table>

### 5.8.1.2 Properties

Properties of the `TDAColumn` class.

For a complete list of the `TDAColumn` class members, see the `TDAColumn Members` topic.

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FieldType</strong></td>
<td>Used to specify the types of values that will be loaded.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Used to specify the field name of loading table.</td>
</tr>
</tbody>
</table>

**See Also**
5.8.1.1.2.1  FieldType Property

Used to specify the types of values that will be loaded.

Class

TDAColumn

Syntax

```property` FieldType: TFieldType `default` ftString;```

Remarks

Use the FieldType property to specify the types of values that will be loaded. Field types for columns may not match data types for the corresponding fields in the database table. TDALoader will cast data values to the types of their fields.

5.8.1.1.2.2  Name Property

Used to specify the field name of loading table.

Class

TDAColumn

Syntax

```property` Name: string;```

Remarks

Each TDAColumn corresponds to one field of the loading table. Use the Name property to specify the name of this field.

See Also
5.8.1.2 TDAColumns Class

Holds a collection of TDAColumn objects.

For a list of all members of this type, see TDAColumns members.

Unit

DALoader

Syntax

TDAColumns = class(TOwnedCollection);

Remarks

Each TDAColumns holds a collection of TDAColumn objects. TDAColumns maintains an index of the columns in its Items array. The Count property contains the number of columns in the collection. At design-time, use the Columns editor to add, remove, or modify columns.

See Also

- TDALoader
- TDAColumn

5.8.1.2.1 Members

TDAColumns class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to access individual columns.</td>
</tr>
</tbody>
</table>
5.8.1.2.2 Properties

Properties of the TDAColumns class.

For a complete list of the TDAColumns class members, see the TDAColumns Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to access individual columns.</td>
</tr>
</tbody>
</table>

See Also

- TDAColumns Class
- TDAColumns Class Members

© 1997-2021 Devart. All Rights Reserved.

5.8.1.2.2.1 Items Property (Indexer)

Used to access individual columns.

Class

TDAColumns

Syntax

```property Items[Index: integer]: TDAColumn; default;```

Parameters

Index

Holds the Index of TDAColumn to refer to.

Remarks

Use the Items property to access individual columns. The value of the Index parameter corresponds to the Index property of TDAColumn.

See Also

- TDAColumn

© 1997-2021 Devart
5.8.1.3 TDALoader Class

This class allows loading external data into database.

For a list of all members of this type, see TDALoader members.

Unit
DALoader

Syntax

TDALoader = class(TComponent);

Remarks

TDALoader allows loading external data into database. To specify the name of loading table set the TDALoader.TableName property. Use the TDALoader.Columns property to access individual columns. Write the TDALoader.OnGetColumnData or TDALoader.OnPutData event handlers to read external data and pass it to the database. Call the TDALoader.Load method to start loading data.

© 1997-2021
Devart. All Rights Reserved.

5.8.1.3.1 Members

TDALoader class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>Used to add a TDAColumn object for each field that will be loaded.</td>
</tr>
<tr>
<td>Connection</td>
<td>property. Used to specify TCustomDAConnection in which TDALoader will be executed.</td>
</tr>
<tr>
<td>TableName</td>
<td>Used to specify the name of the table to which data will be loaded.</td>
</tr>
</tbody>
</table>
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateColumns</td>
<td>Creates <code>TDAColumn</code> objects for all fields of the table with the same name as <code>TDALoader.TableName</code>.</td>
</tr>
<tr>
<td>Load</td>
<td>Starts loading data.</td>
</tr>
<tr>
<td>LoadFromDataSet</td>
<td>Loads data from the specified dataset.</td>
</tr>
<tr>
<td>PutColumnData</td>
<td>Overloaded. Puts the value of individual columns.</td>
</tr>
</tbody>
</table>

## Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnGetColumnData</td>
<td>Occurs when it is needed to put column values.</td>
</tr>
<tr>
<td>OnProgress</td>
<td>Occurs if handling data loading progress of the <code>TDALoader.LoadFromDataSet</code> method is needed.</td>
</tr>
<tr>
<td>OnPutData</td>
<td>Occurs when putting loading data by rows is needed.</td>
</tr>
</tbody>
</table>

## Properties

Properties of the `TDALoader` class.

For a complete list of the `TDALoader` class members, see the [TDALoader Members](#) topic.
### Columns Property

Used to add a **TDAColumn** object for each field that will be loaded.

**Class**

**TDALoader**

**Syntax**

```pascal
property Columns: TDAColumns stored IsColumnsStored;
```

**Remarks**

Use the Columns property to add a **TDAColumn** object for each field that will be loaded.

**See Also**

- [TDAColumns](#)

---

### Connection Property

property. Used to specify TCustomDACConnection in which TDALoader will be executed.

**Class**

**TDALoader**

**Syntax**
**property** Connection: TCustomDAConnection;

**Remarks**

Use the Connection property to specify TCustomDAConnection in which TDALoader will be executed. If Connection is not connected, the Load method calls TCustomDAConnection.Connect.

**See Also**
- TCustomDAConnection

### TableName Property

**Remarks**

Used to specify the name of the table to which data will be loaded.

**Class**

TDALoader

**Syntax**

```property** TableName: string;```

**Remarks**

Set the TableName property to specify the name of the table to which data will be loaded. Add TDAColumn objects to Columns for the fields that are needed to be loaded.

**See Also**
- TDAColumn
- TCustomDAConnection.GetTableNames

5.8.1.3.3 Methods

Methods of the TDALoader class.

For a complete list of the TDALoader class members, see the TDALoader Members topic.
Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateColumns</td>
<td>Creates TDAColumn objects for all fields of the table with the same name as TDALoader.TableName.</td>
</tr>
<tr>
<td>Load</td>
<td>Starts loading data.</td>
</tr>
<tr>
<td>LoadFromDataSet</td>
<td>Loads data from the specified dataset.</td>
</tr>
<tr>
<td>PutColumnData</td>
<td>Overloaded. Puts the value of individual columns.</td>
</tr>
</tbody>
</table>

See Also
- TDALoader Class
- TDALoader Class Members

5.8.1.3.3.1 CreateColumns Method

Creates TDAColumn objects for all fields of the table with the same name as TableName.

Class
TDALoader

Syntax

```delphi
procedure CreateColumns;
```

Remarks

Call the CreateColumns method to create TDAColumn objects for all fields of the table with the same name as TableName. If columns were created before, they will be recreated. You can call CreateColumns from the component popup menu at design-time. After you can customize column loading by setting properties of TDAColumn objects.

See Also
- TDAColumn
- TableName
5.8.1.3.3.2  Load Method

Starts loading data.

Class

TDALoader

Syntax

```
procedure Load; virtual;
```

Remarks

Call the Load method to start loading data. At first it is necessary to create columns and write one of the OnPutData or OnGetColumnData event handlers.

See Also

- OnGetColumnData
- OnPutData

5.8.1.3.3.3  LoadFromDataSet Method

Loads data from the specified dataset.

Class

TDALoader

Syntax

```
procedure LoadFromDataSet(DataSet: TDataSet);
```

Parameters

- `DataSet`
  
  Holds the dataset to load data from.

Remarks
Call the LoadFromDataSet method to load data from the specified dataset. There is no need to create columns and write event handlers for \texttt{OnPutData} and \texttt{OnGetColumnData} before calling this method.

### 5.8.1.3.3.4 PutColumnData Method

Puts the value of individual columns.

**Class**

\texttt{TDALoader}

**Overload List**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{PutColumnData(Col: integer; Row: integer; const Value: variant)}</td>
<td>Puts the value of individual columns by the column index.</td>
</tr>
<tr>
<td>\texttt{PutColumnData(const ColName: string; Row: integer; const Value: variant)}</td>
<td>Puts the value of individual columns by the column name.</td>
</tr>
</tbody>
</table>

**Parameters**

- \texttt{Col}
  - Holds the index of a loading column. The first column has index 0.
- \texttt{Row}
  - Holds the number of loading row. Row starts from 1.
- \texttt{Value}
Holds the column value.

Remarks

Call the PutColumnData method to put the value of individual columns. The Col parameter indicates the index of loading column. The first column has index 0. The Row parameter indicates the number of the loading row. Row starts from 1.

This overloaded method works faster because it searches the right index by its index, not by the index name.

The value of a column should be assigned to the Value parameter.

See Also

- **TDALoader.OnPutData**

Syntax

```pascal
procedure PutColumnData(const ColName: string; Row: integer; const Value: variant); overload;
```

Parameters

- **ColName**
  - Holds the name of a loading column.
- **Row**
  - Holds the number of loading row. Row starts from 1.
- **Value**
  - Holds the column value.
5.8.1.3.4 Events

Events of the **TDALoader** class.

For a complete list of the **TDALoader** class members, see the **TDALoader Members** topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnGetColumnData</strong></td>
<td>Occurs when it is needed to put column values.</td>
</tr>
<tr>
<td><strong>OnProgress</strong></td>
<td>Occurs if handling data loading progress of the <strong>TDALoader.LoadFromData Set</strong> method is needed.</td>
</tr>
<tr>
<td><strong>OnPutData</strong></td>
<td>Occurs when putting loading data by rows is needed.</td>
</tr>
</tbody>
</table>

**See Also**
- [TDALoader Class](#)
- [TDALoader Class Members](#)

© 1997-2021 Devart. All Rights Reserved.

### 5.8.1.3.4.1 OnGetColumnData Event

Occurs when it is needed to put column values.

**Class**

**TDALoader**

**Syntax**

```property
OnGetColumnData: TGetColumnDataEvent;
```

**Remarks**

Write the **OnGetColumnData** event handler to put column values. **TDALoader** calls the **OnGetColumnData** event handler for each column in the loop. Column points to a **TDAColumn** object that corresponds to the current loading column. Use its `Name` or `Index` property to identify what column is loading. The `Row` parameter indicates the current loading record. **TDALoader** increments the `Row` parameter when all the columns of the current
record are loaded. The first row is 1. Set EOF to True to stop data loading. Fill the Value parameter by column values. To start loading call the `Load` method.

Another way to load data is using the `OnPutData` event.

**Example**

This handler loads 1000 rows.

```pascal
procedure TfmMain.GetColumnData(Sender: TObject; Column: TDAColumn; Row: Integer; var Value: Variant; var EOF: Boolean);
begin
  if Row <= 1000 then begin
    case Column.Index of
      0: Value := Row;
      1: Value := Random(100);
      2: Value := Random*100;
      3: Value := 'abc01234567890123456789';
      4: Value := Date;
    else
      Value := Null;
  end;
  else
    EOF := True;
end;
```

**See Also**

- `OnPutData`
- `Load`

© 1997-2021 Devart. All Rights Reserved.
Add a handler to this event if you want to handle data loading progress of the `LoadFromDataSet` method.

See Also

- `LoadFromDataSet`

5.8.1.3.4.3 OnPutData Event

Occurs when putting loading data by rows is needed.

Class

`TDALoader`

Syntax

```delphi
property OnPutData: TDAPutDataEvent;
```

Remarks

Write the OnPutData event handler to put loading data by rows.

Note that rows should be loaded from the first in the ascending order.

To start loading, call the `Load` method.

Example

This handler loads 1000 rows.

```delphi
procedure TfmMain.PutData(Sender: TDALoader);
var
  Count: Integer;
  i: Integer;
begin
  Count := StrToInt(edRows.Text);
  for i := 1 to Count do
    begin
      Sender.PutColumnData(0, i, 1);
      Sender.PutColumnData(1, i, Random(100));
      Sender.PutColumnData(2, i, Random*100);
      Sender.PutColumnData(3, i, 'abc01234567890123456789');
      Sender.PutColumnData(4, i, Date);
    end;
end;
```
5.8.1.4 TDALoaderOptions Class

Allows loading external data into database.

For a list of all members of this type, see TDALoaderOptions members.

Unit

DALoader

Syntax

TDALoaderOptions = class(TPersistent);

5.8.1.4.1 Members

TDALoaderOptions class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseBlankValues</td>
<td>Forces PgDAC to fill the buffer with null values after loading a row to the database.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.
Properties of the **TDALoaderOptions** class.

For a complete list of the **TDALoaderOptions** class members, see the [TDALoaderOptions Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UseBlankValues</strong></td>
<td>Forces PgDAC to fill the buffer with null values after loading a row to the database.</td>
</tr>
</tbody>
</table>

**See Also**
- [TDALoaderOptions Class](#)
- [TDALoaderOptions Class Members](#)

© 1997-2021 Devart. All Rights Reserved.

**5.8.1.4.2.1 UseBlankValues Property**

Forces PgDAC to fill the buffer with null values after loading a row to the database.

**Class**

**TDALoaderOptions**

**Syntax**

```plaintext
property UseBlankValues: boolean default True;
```

**Remarks**

Used to force PgDAC to fill the buffer with null values after loading a row to the database.
5.8.2 Types

Types in the DALoader unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAPutDataEvent</td>
<td>This type is used for the TDALoader.OnPutData event.</td>
</tr>
<tr>
<td>TGetColumnDataEvent</td>
<td>This type is used for the TDALoader.OnGetColumnData event.</td>
</tr>
<tr>
<td>TLoaderProgressEvent</td>
<td>This type is used for the TDALoader.OnProgress event.</td>
</tr>
</tbody>
</table>

5.8.2.1 TDAPutDataEvent Procedure Reference

This type is used for the TDALoader.OnPutData event.

Unit

DALoader

Syntax

TDAPutDataEvent = procedure (Sender: TDALoader) of object;

Parameters

Sender
An object that raised the event.

5.8.2.2 TGetColumnDataEvent Procedure Reference

This type is used for the TDALoader.OnGetColumnData event.

Unit
**DALoader**

**Syntax**

```delphi
TGetColumnDataEvent = procedure (Sender: TObject; Column: TDAColumn; Row: integer; var Value: variant; var IsEOF: boolean) of object;
```

**Parameters**

*Sender*
An object that raised the event.

*Column*
Points to `TDAColumn` object that corresponds to the current loading column.

*Row*
Indicates the current loading record.

*Value*
Holds column values.

*IsEOF*
True, if data loading needs to be stopped.

---

**5.8.2.3 TLoaderProgressEvent Procedure Reference**

This type is used for the `TDALoader.OnProgress` event.

**Unit**

`DALoader`

**Syntax**

```delphi
TLoaderProgressEvent = procedure (Sender: TObject; Percent: integer) of object;
```

**Parameters**

*Sender*
An object that raised the event.

*Percent*
Percentage of the load operation progress.
5.9 **DAScript**

This unit contains the base class for the TPgScript component.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAScript</td>
<td>Makes it possible to execute several SQL statements one by one.</td>
</tr>
<tr>
<td>TDAStatement</td>
<td>This class has attributes and methods for controlling single SQL statement of a script.</td>
</tr>
<tr>
<td>TDAStatements</td>
<td>Holds a collection of TDAStatement objects.</td>
</tr>
</tbody>
</table>

**Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAfterStatementExecuteEvent</td>
<td>This type is used for the TDAScript.AfterExecute event.</td>
</tr>
<tr>
<td>TBeforeStatementExecuteEvent</td>
<td>This type is used for the TDAScript.BeforeExecute event.</td>
</tr>
<tr>
<td>TOnErrorEvent</td>
<td>This type is used for the TDAScript.OnError event.</td>
</tr>
</tbody>
</table>

**Enumerations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TErrorAction</td>
<td>Indicates the action to take when the OnError handler exits.</td>
</tr>
</tbody>
</table>
5.9.1 Classes

Classes in the **DAScript** unit.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAScript</td>
<td>Makes it possible to execute several SQL statements one by one.</td>
</tr>
<tr>
<td>TDASTatement</td>
<td>This class has attributes and methods for controlling single SQL statement of a script.</td>
</tr>
<tr>
<td>TDAScript</td>
<td>Holds a collection of TDASTatement objects.</td>
</tr>
</tbody>
</table>

#### 5.9.1.1 TDAScript Class

Makes it possible to execute several SQL statements one by one.

For a list of all members of this type, see [TDAScript members](#).

**Unit**

**DAScript**

**Syntax**

```
TDAScript = class(TComponent);
```

**Remarks**

Often it is necessary to execute several SQL statements one by one. This can be performed using a lot of components such as [TCustomDASQL](#) descendants. Usually it isn't the best solution. With only one TDAScript descedant component you can execute several SQL statements as one. This sequence of statements is called script. To separate single statements use semicolon (;) or slash (/) and for statements that can contain semicolon, only slash. Note that slash must be the first character in line.

Errors that occur during execution can be processed in the [TDAScript.OnError](#) event handler. By default, on error TDAScript shows exception and continues execution.
5.9.1.1.1 Members

**TDAScript** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Used to specify the connection in which the script will be executed.</td>
</tr>
<tr>
<td>DataSet</td>
<td>Refers to a dataset that holds the result set of query execution.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the script execution and all its parameter values.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>Used to set the delimiter string that separates script statements.</td>
</tr>
<tr>
<td>EndLine</td>
<td>Used to get the current statement last line number in a script.</td>
</tr>
<tr>
<td>EndOffset</td>
<td>Used to get the offset in the last line of the current statement.</td>
</tr>
<tr>
<td>EndPos</td>
<td>Used to get the end position of the current statement.</td>
</tr>
<tr>
<td>Macros</td>
<td>Used to change SQL script text in design- or run-time easily.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to get or set script text.</td>
</tr>
<tr>
<td>StartLine</td>
<td>Used to get the current statement start line number in a script.</td>
</tr>
<tr>
<td>StartOffset</td>
<td>Used to get the offset in the first line of the current statement.</td>
</tr>
</tbody>
</table>
### StartPos

**Description:** Used to get the start position of the current statement in a script.

### Statements

**Description:** Contains a list of statements obtained from the SQL property.

---

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BreakExec</strong></td>
<td>Stops script execution.</td>
</tr>
<tr>
<td><strong>ErrorOffset</strong></td>
<td>Used to get the offset of the statement if the Execute method raised an exception.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>Executes a script.</td>
</tr>
<tr>
<td><strong>ExecuteFile</strong></td>
<td>Executes SQL statements contained in a file.</td>
</tr>
<tr>
<td><strong>ExecuteNext</strong></td>
<td>Executes the next statement in the script and then stops.</td>
</tr>
<tr>
<td><strong>ExecuteStream</strong></td>
<td>Executes SQL statements contained in a stream object.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>MacroByName</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
</tbody>
</table>

---

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong></td>
<td>Occurs after a SQL script execution.</td>
</tr>
<tr>
<td><strong>BeforeExecute</strong></td>
<td>Occurs when taking a specific action before executing the current SQL statement is needed.</td>
</tr>
<tr>
<td><strong>OnError</strong></td>
<td>Occurs when PostgreSQL raises an error.</td>
</tr>
</tbody>
</table>
5.9.1.1.2 Properties

Properties of the **TDAScript** class.

For a complete list of the **TDAScript** class members, see the [TDAScript Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Used to specify the connection in which the script will be executed.</td>
</tr>
<tr>
<td>DataSet</td>
<td>Refers to a dataset that holds the result set of query execution.</td>
</tr>
<tr>
<td>EndLine</td>
<td>Used to get the current statement last line number in a script.</td>
</tr>
<tr>
<td>EndOffset</td>
<td>Used to get the offset in the last line of the current statement.</td>
</tr>
<tr>
<td>EndPos</td>
<td>Used to get the end position of the current statement.</td>
</tr>
<tr>
<td>StartLine</td>
<td>Used to get the current statement start line number in a script.</td>
</tr>
<tr>
<td>StartOffset</td>
<td>Used to get the offset in the first line of the current statement.</td>
</tr>
<tr>
<td>StartPos</td>
<td>Used to get the start position of the current statement in a script.</td>
</tr>
<tr>
<td>Statements</td>
<td>Contains a list of statements obtained from the SQL property.</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>Used to display the script execution and all its parameter values.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>Used to set the delimiter string that separates script</td>
</tr>
</tbody>
</table>
### 5.9.1.1.2.1 Connection Property

Used to specify the connection in which the script will be executed.

**Class**

**TDAScript**

**Syntax**

```plaintext
property Connection: TCustomDAConnection;
```

**Remarks**

Use the Connection property to specify the connection in which the script will be executed. If Connection is not connected, the **Execute** method calls the Connect method of Connection.

Set at design-time by selecting from the list of provided **TCustomDAConnection** objects.

At run-time, set the Connection property to reference an existing TCustomDAConnection object.

**See Also**

- **TCustomDAConnection**

© 1997-2021

Devart. All Rights Reserved.
5.9.1.2.2 Dataset Property

Refers to a dataset that holds the result set of query execution.

Class
TDAScript

Syntax

```property DataSet: TCustomDADataset;
```

Remarks

Set the Dataset property to retrieve the results of the SELECT statements execution inside a script.

See Also
- ExecuteNext
- Execute

© 1997-2021
Devart. All Rights Reserved.

5.9.1.2.3 Debug Property

Used to display the script execution and all its parameter values.

Class
TDAScript

Syntax

```property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

**Note:** If TPgSQLMonitor is used in the project and the TPgSQLMonitor.Active property is set
5.9.1.2.4 Delimiter Property

Used to set the delimiter string that separates script statements.

Class

TDAScript

Syntax

property Delimiter: string stored IsDelimiterStored;

Remarks

Use the Delimiter property to set the delimiter string that separates script statements. By default it is semicolon (;). You can use slash (/) to separate statements that can contain semicolon if the Delimiter property's default value is semicolon. Note that slash must be the first character in line.

5.9.1.2.5 EndLine Property

Used to get the current statement last line number in a script.

Class

TDAScript

Syntax

property EndLine: Int64;

Remarks

Use the EndLine property to get the current statement last line number in a script.
5.9.1.1.2.6  EndOffset Property

Used to get the offset in the last line of the current statement.

Class

TDAScript

Syntax

property EndOffset: Int64;

Remarks

Use the EndOffset property to get the offset in the last line of the current statement.

© 1997-2021
Devart. All Rights Reserved.

5.9.1.1.2.7  EndPos Property

Used to get the end position of the current statement.

Class

TDAScript

Syntax

property EndPos: Int64;

Remarks

Use the EndPos property to get the end position of the current statement (the position of the last character in the statement) in a script.

© 1997-2021
Devart. All Rights Reserved.

5.9.1.1.2.8  Macros Property

Used to change SQL script text in design- or run-time easily.

Class

TDAScript
Syntax

**property** Macros: **TMacros stored** False;

Remarks

With the help of macros you can easily change SQL script text in design- or run-time. Macros extend abilities of parameters and allow changing conditions in the WHERE clause or sort order in the ORDER BY clause. You just insert &MacroName in a SQL query text and change value of macro by the Macro property editor in design-time or the MacroByName function in run-time. In time of opening query macro is replaced by its value.

See Also

- **TMacro**
- **MacroByName**

5.9.1.1.2.9 SQL Property

Used to get or set script text.

Class

**TDAScript**

Syntax

**property** SQL: TStrings;

Remarks

Use the SQL property to get or set script text.

5.9.1.1.2.10 StartLine Property

Used to get the current statement start line number in a script.

Class
**TDAScript**

### Syntax

```fortran
property StartLine: Int64;
```

### Remarks

Use the StartLine property to get the current statement start line number in a script.

© 1997-2021 Devart. All Rights Reserved.

5.9.1.2.11 StartOffset Property

Used to get the offset in the first line of the current statement.

### Class

**TDAScript**

### Syntax

```fortran
property StartOffset: Int64;
```

### Remarks

Use the StartOffset property to get the offset in the first line of the current statement.

© 1997-2021 Devart. All Rights Reserved.

5.9.1.2.12 StartPos Property

Used to get the start position of the current statement in a script.

### Class

**TDAScript**

### Syntax

```fortran
property StartPos: Int64;
```

### Remarks


Use the StartPos property to get the start position of the current statement (the position of the first statement character) in a script.

Contains a list of statements obtained from the SQL property.

**Class**

**TDAScript**

**Syntax**

```
property Statements: TDAStatements;
```

**Remarks**

Contains a list of statements that are obtained from the SQL property. Use the Access Statements property to view SQL statement, set parameters or execute the specified statement. Statements is a zero-based array of statement records. Index specifies the array element to access.

For example, consider the following script:

```sql
CREATE TABLE A (FIELD1 INTEGER);
INSERT INTO A VALUES(1);
INSERT INTO A VALUES(2);
INSERT INTO A VALUES(3);
CREATE TABLE B (FIELD1 INTEGER);
INSERT INTO B VALUES(1);
INSERT INTO B VALUES(2);
INSERT INTO B VALUES(3);
```

**Note**: The list of statements is created and filled when the value of Statements property is requested. That's why the first access to the Statements property can take a long time.

**Example**

You can use the Statements property in the following way:

```delphi
procedure TForm1.Button1Click(Sender: TObject);
var
  i: integer;
begin
  with Script do
  begin
```

© 2021 Devart
for i := 0 to Statements.Count - 1 do
  if Copy(Statements[i].SQL, 1, 6) <> 'CREATE' then
    Statements[i].Execute;
  end;
end;

See Also
- `TDAScript` class

For a complete list of the `TDAScript` class members, see the `TDAScript Members` topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BreakExec</code></td>
<td>Stops script execution.</td>
</tr>
<tr>
<td><code>ErrorOffset</code></td>
<td>Used to get the offset of the statement if the Execute method raised an exception.</td>
</tr>
<tr>
<td><code>Execute</code></td>
<td>Executes a script.</td>
</tr>
<tr>
<td><code>ExecuteFile</code></td>
<td>Executes SQL statements contained in a file.</td>
</tr>
<tr>
<td><code>ExecuteNext</code></td>
<td>Executes the next statement in the script and then stops.</td>
</tr>
<tr>
<td><code>ExecuteStream</code></td>
<td>Executes SQL statements contained in a stream object.</td>
</tr>
<tr>
<td><code>FindMacro</code></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><code>MacroByName</code></td>
<td>Finds a macro with the specified name.</td>
</tr>
</tbody>
</table>

See Also
- `TDAScript Class`
- `TDAScript Class Members`
5.9.1.1.3.1 BreakExec Method

Stops script execution.

Class

TDAScript

Syntax

```pascal
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to stop script execution.

© 1997-2021
Devart. All Rights Reserved.

5.9.1.1.3.2 ErrorOffset Method

Used to get the offset of the statement if the Execute method raised an exception.

Class

TDAScript

Syntax

```pascal
function ErrorOffset: Int64;
```

Return Value

offset of an error.

Remarks

Call the ErrorOffset method to get the offset of the statement if the Execute method raised an exception.

See Also

• OnError

© 1997-2021
Devart. All Rights Reserved.
5.9.1.3.3 Execute Method

Executes a script.

Class

TDAScript

Syntax

procedure Execute; virtual;

Remarks

Call the Execute method to execute a script. If PostgreSQL raises an error, the OnError event occurs.

See Also

- ExecuteNext
- OnError
- ErrorOffset

5.9.1.3.4 ExecuteFile Method

Executes SQL statements contained in a file.

Class

TDAScript

Syntax

procedure ExecuteFile(const FileName: string);

Parameters

FileName

Holds the file name.

Remarks

Call the ExecuteFile method to execute SQL statements contained in a file. Script doesn't load full content into memory. Reading and execution is performed by blocks of 64k size.
Therefore, it is optimal to use it for big files.

5.9.1.3.5 ExecuteNext Method

Executes the next statement in the script and then stops.

Class
TDAScript

Syntax

function ExecuteNext: boolean; virtual;

Return Value
True, if there are any statements left in the script, False otherwise.

Remarks
Use the ExecuteNext method to execute the next statement in the script statement and stop. If PostgreSQL raises an error, the OnError event occurs.

See Also
• Execute
• OnError
• ErrorOffset

5.9.1.3.6 ExecuteStream Method

Executes SQL statements contained in a stream object.

Class
TDAScript

Syntax

procedure ExecuteStream(Stream: TStream);
Parameters
Stream
Holds the stream object from which the statements will be executed.

Remarks
Call the ExecuteStream method to execute SQL statements contained in a stream object.
Reading from the stream and execution is performed by blocks of 64k size.

FindMacro Method

Finds a macro with the specified name.

Class
TDAScript

Syntax
function FindMacro(Name: string): TMacro;

Parameters
Name
Holds the name of a macro to search for.

Return Value
TMacro object if a match is found, nil otherwise.

Remarks
Call the FindMacro method to find a macro with the specified name. If a match is found,
FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct
reference to the TMacros.Items property to avoid depending on the order of the items.

See Also
• TMacro
• Macros
• MacroByName

© 1997-2021
Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback

© 1997-2021
Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback

© 2021 Devart
5.9.1.3.8  MacroByName Method

Finds a macro with the specified name.

Class

TDAScript

Syntax

function MacroByName(Name: string): TMacro;

Parameters

Name

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

See Also

- TMacro
- Macros
- FindMacro

© 1997-2021
Devart. All Rights Reserved.

5.9.1.4  Events

Events of the TDAScript class.

For a complete list of the TDAScript class members, see the TDAScript Members topic.
### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong></td>
<td>Occurs after a SQL script execution.</td>
</tr>
<tr>
<td><strong>BeforeExecute</strong></td>
<td>Occurs when taking a specific action before executing the current SQL statement is needed.</td>
</tr>
<tr>
<td><strong>OnError</strong></td>
<td>Occurs when PostgreSQL raises an error.</td>
</tr>
</tbody>
</table>

### See Also
- TDAScript Class
- TDAScript Class Members

© 1997-2021 Devart. All Rights Reserved.

### 5.9.1.4.1 AfterExecute Event

Occurs after a SQL script execution.

### Class

**TDAScript**

### Syntax

```delphi
property AfterExecute: TAfterStatementExecuteEvent;
```

### Remarks

Occurs after a SQL script has been executed.

### See Also
- Execute

© 1997-2021 Devart. All Rights Reserved.
5.9.1.1.4.2 BeforeExecute Event

Occurs when taking a specific action before executing the current SQL statement is needed.

Class
TDAScript

Syntax

```pascal
property BeforeExecute: TBeforeStatementExecuteEvent;
```

Remarks
Write the BeforeExecute event handler to take specific action before executing the current SQL statement. SQL holds text of the current SQL statement. Write SQL to change the statement that will be executed. Set Omit to True to skip statement execution.

© 1997-2021 Devart. All Rights Reserved.

5.9.1.1.4.3 OnError Event

Occurs when PostgreSQL raises an error.

Class
TDAScript

Syntax

```pascal
property OnError: TOnErrorEvent;
```

Remarks
Occurs when PostgreSQL raises an error.

Action indicates the action to take when the OnError handler exits. On entry into the handler, Action is always set to eaFail.

See Also
- ErrorOffset

© 1997-2021 Devart. All Rights Reserved.
5.9.1.2  TDAStatement Class

This class has attributes and methods for controlling single SQL statement of a script.

For a list of all members of this type, see TDAStatement members.

Unit

DAScript

Syntax

TDAStatement = class(TCollectionItem);

Remarks

TDAScript contains SQL statements, represented as TDAStatement objects. The TDAStatement class has attributes and methods for controlling single SQL statement of a script.

See Also

- TDAScript
- TDAStatements

© 1997-2021
Devart. All Rights Reserved.

5.9.1.2.1  Members

TDAStatement class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndLine</td>
<td>Used to determine the number of the last statement line in a script.</td>
</tr>
<tr>
<td>EndOffset</td>
<td>Used to get the offset in the last line of the statement.</td>
</tr>
<tr>
<td>EndPos</td>
<td>Used to get the end position of the statement in a script.</td>
</tr>
<tr>
<td>Omit</td>
<td>Used to avoid execution of a statement.</td>
</tr>
<tr>
<td>Params</td>
<td>Contains parameters for an</td>
</tr>
</tbody>
</table>
5.9.1.2.2 Properties

Properties of the TDAStatement class.

For a complete list of the TDAStatement class members, see the TDAStatement Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndLine</td>
<td>Used to determine the number of the last statement line in a script.</td>
</tr>
<tr>
<td>EndOffset</td>
<td>Used to get the offset in the last line of the statement.</td>
</tr>
<tr>
<td>EndPos</td>
<td>Used to get the end position of the statement in a script.</td>
</tr>
<tr>
<td>Omit</td>
<td>Used to avoid execution of a statement.</td>
</tr>
<tr>
<td>Params</td>
<td>Contains parameters for an</td>
</tr>
</tbody>
</table>
## EndLine Property

Used to determine the number of the last statement line in a script.

### Class

**TDASTatement**

### Syntax

```
property EndLine: integer;
```

### Remarks

Use the EndLine property to determine the number of the last statement line in a script.
### EndOffset Property

Used to get the offset in the last line of the statement.

**Class**

`TDAStatement`

**Syntax**

```property
EndOffset: integer;
```

**Remarks**

Use the EndOffset property to get the offset in the last line of the statement.

© 1997-2021 Devart. All Rights Reserved.

### EndPos Property

Used to get the end position of the statement in a script.

**Class**

`TDAStatement`

**Syntax**

```property
EndPos: integer;
```

**Remarks**

Use the EndPos property to get the end position of the statement (the position of the last character in the statement) in a script.

© 1997-2021 Devart. All Rights Reserved.

### Omit Property

Used to avoid execution of a statement.

**Class**

`TDAStatement`
### Syntax

**property** Omit: boolean;

### Remarks

Set the Omit property to True to avoid execution of a statement.

5.9.1.2.2.5 Params Property

Contains parameters for an SQL statement.

### Class

**TDASTatement**

### Syntax

**property** Params: TDAParams;

### Remarks

Contains parameters for an SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically. Params is a zero-based array of parameter records. Index specifies the array element to access.

### See Also

- **TDAParam**

5.9.1.2.2.6 Script Property

Used to determine the TDAScript object the SQL Statement belongs to.

### Class

**TDASTatement**
5.9.1.2.2.7 SQL Property

Used to get or set the text of an SQL statement.

Class

TDAScript

Syntax

<table>
<thead>
<tr>
<th>property</th>
<th>SQL: string</th>
</tr>
</thead>
</table>

Remarks

Use the SQL property to get or set the text of an SQL statement.

5.9.1.2.2.8 StartLine Property

Used to determine the number of the first statement line in a script.

Class

TDAScript

Syntax

<table>
<thead>
<tr>
<th>property</th>
<th>StartLine: integer</th>
</tr>
</thead>
</table>

Remarks

Use the StartLine property to determine the number of the first statement line in a script.
5.9.1.2.2.9 StartOffset Property

Used to get the offset in the first line of a statement.

**Class**

`TDAStratement`

**Syntax**

```property
startOffset: integer;
```

**Remarks**

Use the StartOffset property to get the offset in the first line of a statement.

5.9.1.2.2.10 StartPos Property

Used to get the start position of the statement in a script.

**Class**

`TDAStratement`

**Syntax**

```property
StartPos: integer;
```

**Remarks**

Use the StartPos property to get the start position of the statement (the position of the first statement character) in a script.

5.9.1.2.3 Methods

Methods of the `TDAStratement` class.

For a complete list of the `TDAStratement` class members, see the `TDAStratement Members`
5.9.1.2.3.1 Execute Method

Executes a statement.

Class

TDAStatement

Syntax

```
procedure Execute;
```

Remarks

Use the Execute method to execute a statement.

© 1997-2021
Devart. All Rights Reserved.

5.9.1.3 TDAStatements Class

Holds a collection of TDAStatement objects.

For a list of all members of this type, see TDAStatements members.

Unit

DAScript

Syntax
TDAStatements = class(TCollection);

Remarks

Each TDAStatements holds a collection of TDAStatement objects. TDAStatements maintains an index of the statements in its Items array. The Count property contains the number of statements in the collection. Use TDAStatements class to manipulate script SQL statements.

See Also

- TDAScript
- TDAStatement

© 1997-2021 Devart. All Rights Reserved.

5.9.1.3.1 Members

**TDAStatements** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to access separate script statements.</td>
</tr>
</tbody>
</table>

5.9.1.3.2 Properties

Properties of the **TDAStatements** class.

For a complete list of the **TDAStatements** class members, see the **TDAStatements Members** topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to access separate script statements.</td>
</tr>
</tbody>
</table>

See Also
5.9.1.3.2.1 Items Property (Indexer)

Used to access separate script statements.

Class

TDAStatements

Syntax

```delphi
property Items[Index: Integer]: TDAStatement; default;
```

Parameters

- **Index**
  
  Holds the index value.

Remarks

Use the Items property to access individual script statements. The value of the Index parameter corresponds to the Index property of `TDAStatement`.

See Also

- `TDAStatement`

Types

Types in the `DAScript` unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TAfterStatementExecuteEvent</code></td>
<td>This type is used for the <code>TDAScript.AfterExecute</code> event.</td>
</tr>
<tr>
<td><code>TBeforeStatementExecuteEvent</code></td>
<td>This type is used for the <code>TDAScript.BeforeExecute</code> event.</td>
</tr>
</tbody>
</table>
5.9.2.1  **TAfterStatementExecuteEvent Procedure Reference**

This type is used for the **TDAScript.AfterExecute** event.

**Unit**

**DAScript**

**Syntax**

\[
\text{TAfterStatementExecuteEvent} = \text{procedure} \ (\text{Sender: TObject; SQL: string}) \ \text{of object};
\]

**Parameters**

- **Sender**
  - An object that raised the event.
- **SQL**
  - Holds the passed SQL statement.

---

5.9.2.2  **TBeforeStatementExecuteEvent Procedure Reference**

This type is used for the **TDAScript.BeforeExecute** event.

**Unit**

**DAScript**

**Syntax**

\[
\text{TBeforeStatementExecuteEvent} = \text{procedure} \ (\text{Sender: TObject; \text{var SQL: string; \text{var Omit: boolean}) of object}};
\]

**Parameters**

- **Sender**
An object that raised the event.

**SQL**
Holds the passed SQL statement.

**Omit**
True, if the statement execution should be skipped.

5.9.2.3 **TOnErrorEvent Procedure Reference**

This type is used for the `TDAScript.OnError` event.

**Unit**
`DAScript`

**Syntax**

```
TOnErrorEvent = procedure (Sender: TObject; E: Exception; SQL: string; var Action: TErrorAction) of object;
```

**Parameters**

**Sender**
An object that raised the event.

**E**
The error code.

**SQL**
Holds the passed SQL statement.

**Action**
The action to take when the OnError handler exits.

5.9.3 **Enumerations**

Enumerations in the `DAScript` unit.

**Enumerations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TErrorAction</code></td>
<td>Indicates the action to take when the OnError handler</td>
</tr>
</tbody>
</table>
5.9.3.1 TErrorAction Enumeration

Indicates the action to take when the OnError handler exits.

Unit

DAScript

Syntax

TErrorAction = (eaAbort, eaFail, eaException, eaContinue);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>eaAbort</td>
<td>Abort execution without displaying an error message.</td>
</tr>
<tr>
<td>eaContinue</td>
<td>Continue execution.</td>
</tr>
<tr>
<td>eaException</td>
<td>In Delphi 6 and higher exception is handled by the Application.HandleException method.</td>
</tr>
<tr>
<td>eaFail</td>
<td>Abort execution and display an error message.</td>
</tr>
</tbody>
</table>

5.10 DASQLMonitor

This unit contains the base class for the TPgSQLMonitor component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomDASQLMonitor</td>
<td>A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.</td>
</tr>
<tr>
<td>TDBMonitorOptions</td>
<td>This class holds options for dbMonitor.</td>
</tr>
</tbody>
</table>
Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATraceFlags</td>
<td>Represents the set of TDATraceFlag.</td>
</tr>
<tr>
<td>TMonitorOptions</td>
<td>Represents the set of TMonitorOption.</td>
</tr>
<tr>
<td>TOnSQLEvent</td>
<td>This type is used for the TCustomDASQLMonitor.On SQL event.</td>
</tr>
</tbody>
</table>

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATraceFlag</td>
<td>Use TraceFlags to specify which database operations the monitor should track in an application at runtime.</td>
</tr>
<tr>
<td>TMonitorOption</td>
<td>Used to define where information from SQLMonitor will be dispayed.</td>
</tr>
</tbody>
</table>

5.10.1 Classes

Classes in the DASQLMonitor unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomDASQLMonitor</td>
<td>A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.</td>
</tr>
<tr>
<td>TDBMonitorOptions</td>
<td>This class holds options for dbMonitor.</td>
</tr>
</tbody>
</table>
5.10.1.1 TCustomDASQLMonitor Class

A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.

For a list of all members of this type, see TCustomDASQLMonitor members.

Unit
DASQLMonitor

Syntax

```
TCustomDASQLMonitor = class(TComponent);
```

Remarks

TCustomDASQLMonitor is a base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively. TCustomDASQLMonitor provides two ways of displaying debug information. It monitors either by dialog window or by Borland's proprietary SQL Monitor. Furthermore to receive debug information use the TCustomDASQLMonitor.OnSQL event.

In applications use descendants of TCustomDASQLMonitor.

5.10.1.1.1 Members

**TCustomDASQLMonitor** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to activate monitoring of SQL.</td>
</tr>
<tr>
<td>DBMonitorOptions</td>
<td>Used to set options for dbMonitor.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to include the desired properties for TCustomDASQLMonitor.</td>
</tr>
<tr>
<td>TraceFlags</td>
<td>Used to specify which</td>
</tr>
</tbody>
</table>
database operations the monitor should track in an application at runtime.

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSQL</td>
<td>Occurs when tracing of SQL activity on database components is needed.</td>
</tr>
</tbody>
</table>

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to activate monitoring of SQL.</td>
</tr>
<tr>
<td>DBMonitorOptions</td>
<td>Used to set options for dbMonitor.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to include the desired properties for TCustomDASQLMonitor.</td>
</tr>
<tr>
<td>TraceFlags</td>
<td>Used to specify which database operations the monitor should track in an application at runtime.</td>
</tr>
</tbody>
</table>

See Also

- TCustomDASQLMonitor Class
- TCustomDASQLMonitor Class Members
5.10.1.1.2.1 Active Property

Used to activate monitoring of SQL.

Class

TCustomDASQLMonitor

Syntax

```pascal
property Active: boolean default True;
```

Remarks

Set the Active property to True to activate monitoring of SQL.

See Also

- OnSQL

© 1997-2021

Devart. All Rights Reserved.

5.10.1.1.2.2 DBMonitorOptions Property

Used to set options for dbMonitor.

Class

TCustomDASQLMonitor

Syntax

```pascal
property DBMonitorOptions: TDBMonitorOptions;
```

Remarks

Use DBMonitorOptions to set options for dbMonitor.

© 1997-2021

Devart. All Rights Reserved.

5.10.1.1.2.3 Options Property

Used to include the desired properties for TCustomDASQLMonitor.

Class
**TCustomDASQLMonitor**

**Syntax**

```plaintext
property Options: TMonitorOptions default [moDialog,
moSQLMonitor, moDBMonitor, moCustom];
```

**Remarks**

Set Options to include the desired properties for TCustomDASQLMonitor.

**See Also**

- [OnSQL](#)

© 1997-2021 Devart. All Rights Reserved.

---

**5.10.1.1.2.4 TraceFlags Property**

Used to specify which database operations the monitor should track in an application at runtime.

**Class**

**TCustomDASQLMonitor**

**Syntax**

```plaintext
property TraceFlags: TDATraceFlags default [tfQPrepare,
tfQExecute, tfError, tfConnect, tfTransact, tfParams, tfMisc];
```

**Remarks**

Use the TraceFlags property to specify which database operations the monitor should track in an application at runtime.

**See Also**

- [OnSQL](#)

© 1997-2021 Devart. All Rights Reserved.
5.10.1.1.3 Events

Events of the **TCustomDASQLMonitor** class.

For a complete list of the **TCustomDASQLMonitor** class members, see the **TCustomDASQLMonitor Members** topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSQL</td>
<td>Occurs when tracing of SQL activity on database components is needed.</td>
</tr>
</tbody>
</table>

**See Also**

- **TCustomDASQLMonitor Class**
- **TCustomDASQLMonitor Class Members**

© 1997-2021 Devart. All Rights Reserved.

5.10.1.3.1 OnSQL Event

Occurs when tracing of SQL activity on database components is needed.

**Class**

**TCustomDASQLMonitor**

**Syntax**

```
property OnSQL: TOnSQLEvent;
```

**Remarks**

Write the OnSQL event handler to let an application trace SQL activity on database components. The Text parameter holds the detected SQL statement. Use the Flag parameter to make selective processing of SQL in the handler body.

**See Also**

- **TraceFlags**

© 1997-2021 Devart. All Rights Reserved.
5.10.1.2 TDBMonitorOptions Class

This class holds options for dbMonitor.

For a list of all members of this type, see TDBMonitorOptions members.

Unit

DASQLMonitor

Syntax

TDBMonitorOptions = class(TPersistent);

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Used to set the host name or IP address of the computer where dbMonitor application runs.</td>
</tr>
<tr>
<td>Port</td>
<td>Used to set the port number for connecting to dbMonitor.</td>
</tr>
<tr>
<td>ReconnectTimeout</td>
<td>Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.</td>
</tr>
<tr>
<td>SendTimeout</td>
<td>Used to set timeout for sending events to dbMonitor.</td>
</tr>
</tbody>
</table>
5.10.1.2.2 Properties

Properties of the **TDBMonitorOptions** class.

For a complete list of the **TDBMonitorOptions** class members, see the [TDBMonitorOptions Members](#) topic.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Host</strong></td>
<td>Used to set the host name or IP address of the computer where dbMonitor application runs.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Used to set the port number for connecting to dbMonitor.</td>
</tr>
<tr>
<td><strong>ReconnectTimeout</strong></td>
<td>Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.</td>
</tr>
<tr>
<td><strong>SendTimeout</strong></td>
<td>Used to set timeout for sending events to dbMonitor.</td>
</tr>
</tbody>
</table>

### See Also
- [TDBMonitorOptions Class](#)
- [TDBMonitorOptions Class Members](#)

© 1997-2021 Devart. All Rights Reserved.  
[Request Support]  [DAC Forum]  [Provide Feedback]

5.10.1.2.2.1 Host Property

Used to set the host name or IP address of the computer where dbMonitor application runs.

**Class**

**TDBMonitorOptions**

**Syntax**

```property Host: string;```

**Remarks**
Use the Host property to set the host name or IP address of the computer where dbMonitor application runs.

dbMonitor supports remote monitoring. You can run dbMonitor on a different computer than monitored application runs. In this case you need to set the Host property to the corresponding computer name.

5.10.1.2.2.2 Port Property

Used to set the port number for connecting to dbMonitor.

Class

TDBMonitorOptions

Syntax

```property` Port: integer `default` DBMonitorPort;```

Remarks

Use the Port property to set the port number for connecting to dbMonitor.

5.10.1.2.2.3 ReconnectTimeout Property

Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.

Class

TDBMonitorOptions

Syntax

```property` ReconnectTimeout: integer `default` DefaultReconnectTimeout;```

Remarks
Use the ReconnectTimeout property to set the minimum time (in milliseconds) that should be spent before allowing reconnecting to dbMonitor. If an error occurs when the component sends an event to dbMonitor (dbMonitor is not running), next events are ignored and the component does not restore the connection until ReconnectTimeout is over.

5.10.1.2.2.4 SendTimeout Property

Used to set timeout for sending events to dbMonitor.

Class

**TDBMonitorOptions**

Syntax

```plaintext
property SendTimeout: integer default DefaultSendTimeout;
```

Remarks

Use the SendTimeout property to set timeout (in milliseconds) for sending events to dbMonitor. If dbMonitor does not respond in the specified timeout, event is ignored.

Types

Types in the **DASQLMonitor** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TDATraceFlags</strong></td>
<td>Represents the set of <strong>TDATraceFlag</strong>.</td>
</tr>
<tr>
<td><strong>TMonitorOptions</strong></td>
<td>Represents the set of <strong>TMonitorOption</strong>.</td>
</tr>
<tr>
<td><strong>TOnSQLEvent</strong></td>
<td>This type is used for the <strong>TCustomDASQLMonitor.On SQL</strong> event.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback

© 2021 Devart
5.10.2.1 TDATraceFlags Set

Represents the set of TDATraceFlag.

Unit
DASQLMonitor

Syntax

TDATraceFlags = set of TDATraceFlag;

5.10.2.2 TMonitorOptions Set

Represents the set of TMonitorOption.

Unit
DASQLMonitor

Syntax

TMonitorOptions = set of TMonitorOption;

5.10.2.3 TOnSQLEvent Procedure Reference

This type is used for the TCustomDASQLMonitor.OnSQL event.

Unit
DASQLMonitor

Syntax

TOnSQLEvent = procedure (Sender: TObject; Text: string; Flag: TDATraceFlag) of object;

Parameters
**Sender**
An object that raised the event.

**Text**
Holds the detected SQL statement.

**Flag**
Use the Flag parameter to make selective processing of SQL in the handler body.

## 5.10.3 Enumerations

Enumerations in the **DASQLMonitor** unit.

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATraceFlag</td>
<td>Use TraceFlags to specify which database operations the monitor should track in an application at runtime.</td>
</tr>
<tr>
<td>TMonitorOption</td>
<td>Used to define where information from SQLMonitor will be displayed.</td>
</tr>
</tbody>
</table>

### 5.10.3.1 TDATraceFlag Enumeration

Use TraceFlags to specify which database operations the monitor should track in an application at runtime.

**Unit**

**DASQLMonitor**

**Syntax**

```pascal
tDATraceFlag = (tfQPrepare, tfQExecute, tfQFetch, tfError, tfStmt, tfConnect, tfTransact, tfBlob, tfService, tfMisc, tfParams, tfObjDestroy, tfPool);
```
### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tfBlob</td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td>tfConnect</td>
<td>Establishing a connection.</td>
</tr>
<tr>
<td>tfError</td>
<td>Errors of query execution.</td>
</tr>
<tr>
<td>tfMisc</td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td>tfObjDestroy</td>
<td>Destroying of components.</td>
</tr>
<tr>
<td>tfParams</td>
<td>Representing parameter values for tfQPrepare and tfQExecute.</td>
</tr>
<tr>
<td>tfPool</td>
<td>Connection pool operations.</td>
</tr>
<tr>
<td>tfQExecute</td>
<td>Execution of the queries.</td>
</tr>
<tr>
<td>tfQFetch</td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td>tfQPrepare</td>
<td>Queries preparation.</td>
</tr>
<tr>
<td>tfService</td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td>tfStmt</td>
<td>This option is declared for future use.</td>
</tr>
<tr>
<td>tfTransact</td>
<td>Processing transactions.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  

#### 5.10.3.2 TMonitorOption Enumeration

Used to define where information from SQLMonitor will be displayed.

**Unit**  
DASQLMonitor

**Syntax**

```plaintext
TMonitorOption = (moDialog, moSQLMonitor, moDBMonitor, moCustom, moHandled);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>moCustom</td>
<td>Monitoring of SQL for individual components is allowed. Set Debug</td>
</tr>
<tr>
<td></td>
<td>properties in SQL-related components to True to let TCustomDASQLMonitor</td>
</tr>
<tr>
<td></td>
<td>instance to monitor their behavior. Has effect when moDialog is included.</td>
</tr>
<tr>
<td>moDBMonitor</td>
<td>Debug information is displayed in DBMonitor.</td>
</tr>
</tbody>
</table>
### 5.11 DBAccess

This unit contains base classes for most of the components.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDAError</td>
<td>A base class for exceptions that are raised when an error occurs on the server side.</td>
</tr>
<tr>
<td>TCRDataSource</td>
<td>Provides an interface between a DAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td>TCustomConnectDialog</td>
<td>A base class for the connect dialog components.</td>
</tr>
<tr>
<td>TCustomDAConnection</td>
<td>A base class for components used to establish connections.</td>
</tr>
<tr>
<td>TCustomDADataset</td>
<td>Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.</td>
</tr>
<tr>
<td>TCustomDASQL</td>
<td>A base class for components executing SQL statements that do not return result sets.</td>
</tr>
<tr>
<td>TCustomDAUpdateSQL</td>
<td>A base class for components that provide DML statements for more flexible control over data modifications.</td>
</tr>
<tr>
<td>TDACCondition</td>
<td>Represents a condition from the TDACConditions list.</td>
</tr>
<tr>
<td>TDACConditions</td>
<td>Holds a collection of</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TDACCondition</strong></td>
<td>TDACCondition objects.</td>
</tr>
<tr>
<td><strong>TDACConnectionOptions</strong></td>
<td>This class allows setting up the behaviour of the TDACConnection class.</td>
</tr>
<tr>
<td><strong>TDACConnectionSSLOptions</strong></td>
<td>This class is used to set up the SSL options.</td>
</tr>
<tr>
<td><strong>TDADataSetOptions</strong></td>
<td>This class allows setting up the behaviour of the TDADataSet class.</td>
</tr>
<tr>
<td><strong>TDAEncryption</strong></td>
<td>Used to specify the options of the data encryption in a dataset.</td>
</tr>
<tr>
<td><strong>TDAMapRule</strong></td>
<td>Class that forms rules for Data Type Mapping.</td>
</tr>
<tr>
<td><strong>TDAMapRules</strong></td>
<td>Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.</td>
</tr>
<tr>
<td><strong>TDAMetaData</strong></td>
<td>A class for retrieving metainformation of the specified database objects in the form of dataset.</td>
</tr>
<tr>
<td><strong>TDAParam</strong></td>
<td>A class that forms objects to represent the values of the parameters set.</td>
</tr>
<tr>
<td><strong>TDAParams</strong></td>
<td>This class is used to manage a list of TDAParam objects for an object that uses field parameters.</td>
</tr>
<tr>
<td><strong>TDATransaction</strong></td>
<td>A base class that implements functionality for controlling transactions.</td>
</tr>
<tr>
<td><strong>TMacro</strong></td>
<td>Object that represents the value of a macro.</td>
</tr>
<tr>
<td><strong>TMacro</strong></td>
<td>Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDAODataSet components.</td>
</tr>
<tr>
<td><strong>TPoolingOptions</strong></td>
<td>This class allows setting up the behaviour of the connection pool.</td>
</tr>
<tr>
<td><strong>TSmartFetchOptions</strong></td>
<td>Smart fetch options are</td>
</tr>
</tbody>
</table>
Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAfterExecuteEvent</td>
<td>This type is used for the TCustomDADataSet.AfterExecute and TCustomDASQL.AfterExecute events.</td>
</tr>
<tr>
<td>TAfterFetchEvent</td>
<td>This type is used for the TCustomDADataSet.AfterExecute event.</td>
</tr>
<tr>
<td>TBeforeFetchEvent</td>
<td>This type is used for the TCustomDADataSet.BeforeFetch event.</td>
</tr>
<tr>
<td>TConnectionLostEvent</td>
<td>This type is used for the TCustomDACConnection.OnConnectionLost event.</td>
</tr>
<tr>
<td>TDACConnectionString</td>
<td>This type is used for the TCustomDACConnection.OnError event.</td>
</tr>
<tr>
<td>TDATransactionErrorEvent</td>
<td>This type is used for the TDATransaction.onError event.</td>
</tr>
<tr>
<td>TRefreshOptions</td>
<td>Represents the set of TRefreshOption.</td>
</tr>
<tr>
<td>TUpdateExecuteEvent</td>
<td>This type is used for the TCustomDADataSet.AfterUpdateExecute and TCustomDADataSet.BeforeUpdateExecute events.</td>
</tr>
</tbody>
</table>

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLabelSet</td>
<td>Sets the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>TLockMode</td>
<td>Specifies the lock mode.</td>
</tr>
<tr>
<td>TRefreshOption</td>
<td>Indicates when the editing record will be refreshed.</td>
</tr>
</tbody>
</table>
### Classes

Classes in the **DBAccess** unit.

#### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EDAError</strong></td>
<td>A base class for exceptions that are raised when an error occurs on the server side.</td>
</tr>
<tr>
<td><strong>TCRDataSource</strong></td>
<td>Provides an interface between a DAC dataset.</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>TCustomConnectDialog</td>
<td>A base class for the connect dialog components.</td>
</tr>
<tr>
<td>TCustomDAConnection</td>
<td>A base class for components used to establish connections.</td>
</tr>
<tr>
<td>TCustomDADataset</td>
<td>Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.</td>
</tr>
<tr>
<td>TCustomDASQL</td>
<td>A base class for components executing SQL statements that do not return result sets.</td>
</tr>
<tr>
<td>TCustomDAUpdateSQL</td>
<td>A base class for components that provide DML statements for more flexible control over data modifications.</td>
</tr>
<tr>
<td>TDACCondition</td>
<td>Represents a condition from the TDACConditions list.</td>
</tr>
<tr>
<td>TDACConditions</td>
<td>Holds a collection of TDACCondition objects.</td>
</tr>
<tr>
<td>TDACConnectionOptions</td>
<td>This class allows setting up the behaviour of the TDACConnection class.</td>
</tr>
<tr>
<td>TDACConnectionSSLOptions</td>
<td>This class is used to set up the SSL options.</td>
</tr>
<tr>
<td>TDADatasetOptions</td>
<td>This class allows setting up the behaviour of the TDADataset class.</td>
</tr>
<tr>
<td>TDAEncryption</td>
<td>Used to specify the options of the data encryption in a dataset.</td>
</tr>
<tr>
<td>TDAMapRule</td>
<td>Class that forms rules for Data Type Mapping.</td>
</tr>
<tr>
<td>TDAMapRules</td>
<td>Used for adding rules for Dataset fields mapping with both identifying by field name and by field type and Delphi field types.</td>
</tr>
<tr>
<td>TDAMetaData</td>
<td>A class for retrieving</td>
</tr>
</tbody>
</table>
5.11.1.1 EDAError Class

A base class for exceptions that are raised when an error occurs on the server side.

For a list of all members of this type, see EDAError members.

Unit

DBAccess

Syntax

EDAError = class(EDatabaseError);

Remarks
EDAError is a base class for exceptions that are raised when an error occurs on the server side.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Contains the component that caused the error.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>Determines the error code returned by the server.</td>
</tr>
</tbody>
</table>

See Also

- EDAError Class
- EDAError Class Members
5.11.1.2.1  Component Property

Contains the component that caused the error.

Class

EDAError

Syntax

| property | Component: TObject; |

Remarks

The Component property contains the component that caused the error.

5.11.1.2.2  ErrorCode Property

Determines the error code returned by the server.

Class

EDAError

Syntax

| property | ErrorCode: integer; |

Remarks

Use the ErrorCode property to determine the error code returned by PostgreSQL. This value is always positive.

5.11.2  TCRDataSource Class

Provides an interface between a DAC dataset components and data-aware controls on a form.

For a list of all members of this type, see TCRDataSource members.

© 2021 Devart
Unit

DBAccess

Syntax

TCRDataSource = class(TDataSource);

Remarks

TCRDataSource provides an interface between a DAC dataset components and data-aware controls on a form.

TCRDataSource inherits its functionality directly from the TDataSource component.

At design time assign individual data-aware components' DataSource properties from their drop-down listboxes.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.2.1 Members

TCRDataSource class overview.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.3 TCustomConnectDialog Class

A base class for the connect dialog components.

For a list of all members of this type, see TCustomConnectDialog members.

Unit

DBAccess

Syntax

TCustomConnectDialog = class(TComponent);

Remarks

TCustomConnectDialog is a base class for the connect dialog components. It provides functionality to show a dialog box where user can edit username, password and server name.
before connecting to a database. You can customize captions of buttons and labels by their properties.

© 1997-2021 Devart. All Rights Reserved.

5.11.3.1 Members

**TCustomConnectDialog** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelButton</td>
<td>Used to specify the label for the Cancel button.</td>
</tr>
<tr>
<td>Caption</td>
<td>Used to set the caption of dialog box.</td>
</tr>
<tr>
<td>ConnectButton</td>
<td>Used to specify the label for the Connect button.</td>
</tr>
<tr>
<td>DialogClass</td>
<td>Used to specify the class of the form that will be displayed to enter login information.</td>
</tr>
<tr>
<td>LabelSet</td>
<td>Used to set the language of buttons and labels captions.</td>
</tr>
<tr>
<td>PasswordLabel</td>
<td>Used to specify a prompt for password edit.</td>
</tr>
<tr>
<td>Retries</td>
<td>Used to indicate the number of retries of failed connections.</td>
</tr>
<tr>
<td>SavePassword</td>
<td>Used for the password to be displayed in ConnectDialog in asterisks.</td>
</tr>
<tr>
<td>ServerLabel</td>
<td>Used to specify a prompt for the server name edit.</td>
</tr>
<tr>
<td>StoreLogInfo</td>
<td>Used to specify whether the login information should be kept in system registry after a connection was established.</td>
</tr>
<tr>
<td>UsernameLabel</td>
<td>Used to specify a prompt for username edit.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.</td>
</tr>
<tr>
<td>GetServerList</td>
<td>Retrieves a list of available server names.</td>
</tr>
</tbody>
</table>

Properties of the TCustomConnectDialog class.

For a complete list of the TCustomConnectDialog class members, see the TCustomConnectDialog Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelButton</td>
<td>Used to specify the label for the Cancel button.</td>
</tr>
<tr>
<td>Caption</td>
<td>Used to set the caption of dialog box.</td>
</tr>
<tr>
<td>ConnectButton</td>
<td>Used to specify the label for the Connect button.</td>
</tr>
<tr>
<td>DialogClass</td>
<td>Used to specify the class of the form that will be displayed to enter login information.</td>
</tr>
<tr>
<td>LabelSet</td>
<td>Used to set the language of buttons and labels captions.</td>
</tr>
<tr>
<td>PasswordLabel</td>
<td>Used to specify a prompt for password edit.</td>
</tr>
<tr>
<td>Retries</td>
<td>Used to indicate the number of retries of failed connections.</td>
</tr>
<tr>
<td>SavePassword</td>
<td>Used for the password to be displayed in ConnectDialog in asterisks.</td>
</tr>
</tbody>
</table>
### ServerLabel

Used to specify a prompt for the server name edit.

### StoreLogInfo

Used to specify whether the login information should be kept in system registry after a connection was established.

### UsernameLabel

Used to specify a prompt for username edit.

---

**See Also**

- `TCustomConnectDialog` Class
- `TCustomConnectDialog` Class Members

---

5.11.1.3.2.1 CancelButton Property

**Used to specify the label for the Cancel button.**

**Class**

`TCustomConnectDialog`

**Syntax**

```python
property CancelButton: string;
```

**Remarks**

Use the CancelButton property to specify the label for the Cancel button.

---

5.11.1.3.2.2 Caption Property

**Used to set the caption of dialog box.**

**Class**

`TCustomConnectDialog`
Syntax

```plaintext
property Caption: string;
```

Remarks

Use the Caption property to set the caption of dialog box.

5.11.1.3.2.3 ConnectButton Property

- Used to specify the label for the Connect button.

Class

```
TCustomConnectDialog
```

Syntax

```plaintext
property ConnectButton: string;
```

Remarks

Use the ConnectButton property to specify the label for the Connect button.

5.11.1.3.2.4 DialogClass Property

- Used to specify the class of the form that will be displayed to enter login information.

Class

```
TCustomConnectDialog
```

Syntax

```plaintext
property DialogClass: string;
```

Remarks

Use the DialogClass property to specify the class of the form that will be displayed to enter login information. When this property is blank, TCustomConnectDialog uses the default form -
TConnectForm. You can write your own login form to enter login information and assign its class name to the DialogClass property. Each login form must have ConnectDialog: TCustomConnectDialog published property to access connection information. For details see the implementation of the connect form which sources are in the Lib subdirectory of the PgDAC installation directory.

See Also
- GetServerList

5.11.1.3.2.5 LabelSet Property

Used to set the language of buttons and labels captions.

Class
- TCustomConnectDialog

Syntax

property LabelSet: TLabelSet default lsEnglish;

Remarks

Use the LabelSet property to set the language of labels and buttons captions.

The default value is lsEnglish.

5.11.1.3.2.6 PasswordLabel Property

Used to specify a prompt for password edit.

Class
- TCustomConnectDialog

Syntax

property PasswordLabel: string;
Remarks

Use the PasswordLabel property to specify a prompt for password edit.

Class

TCustomConnectDialog

Syntax

```plaintext
property Retries: word default 3;
```

Remarks

Use the Retries property to determine the number of retries of failed connections.

Class

TCustomConnectDialog

Syntax

```plaintext
property SavePassword: boolean default False;
```

Remarks

If True, and the Password property of the connection instance is assigned, the password in ConnectDialog is displayed in asterisks.
5.11.1.3.2.9 ServerLabel Property

Used to specify a prompt for the server name edit.

Class

TCustomConnectDialog

Syntax

```property ServerLabel: string;```

Remarks

Use the ServerLabel property to specify a prompt for the server name edit.

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

5.11.1.3.2.10 StoreLogInfo Property

Used to specify whether the login information should be kept in system registry after a connection was established.

Class

TCustomConnectDialog

Syntax

```property StoreLogInfo: boolean default True;```

Remarks

Use the StoreLogInfo property to specify whether to keep login information in system registry after a connection was established using provided username, password and servername.

Set this property to True to store login information.

The default value is True.

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

© 2021 Devart
5.11.1.3.2.11 UsernameLabel Property

Used to specify a prompt for username edit.

Class

TCustomConnectDialog

Syntax

property UsernameLabel: string;

Remarks

Use the UsernameLabel property to specify a prompt for username edit.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.</td>
</tr>
<tr>
<td>GetServerList</td>
<td>Retrieves a list of available server names.</td>
</tr>
</tbody>
</table>

See Also

- TCustomConnectDialog Class
- TCustomConnectDialog Class Members

© 1997-2021
Devart. All Rights Reserved.
5.11.1.3.3.1 Execute Method

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.

Class

TCustomConnectDialog

Syntax

function Execute: boolean; virtual;

Return Value

True, if connected.

Remarks

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button. Returns True if connected. If user clicks Cancel, Execute returns False.

In the case of failed connection Execute offers to connect repeat Retries times.

5.11.1.3.3.2 GetServerList Method

Retrieves a list of available server names.

Class

TCustomConnectDialog

Syntax

procedure GetServerList(List: TStrings); virtual;

Parameters

List

Holds a list of available server names.

Remarks

Call the GetServerList method to retrieve a list of available server names. It is particularly relevant for writing custom login form.
5.11.1.4 TCustomDAConnection Class

A base class for components used to establish connections.

For a list of all members of this type, see TCustomDAConnection members.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectDialog</td>
<td>Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td>ConnectString</td>
<td>Used to specify the connection information, such as: UserName, Password,</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ConvertEOL</strong></td>
<td>Allows customizing line breaks in string fields and parameters.</td>
</tr>
<tr>
<td><strong>InTransaction</strong></td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong></td>
<td>Specifies whether a login dialog appears immediately before opening a new connection.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Specifies the connection behavior.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Serves to supply a password for login.</td>
</tr>
<tr>
<td><strong>Pooling</strong></td>
<td>Enables or disables using connection pool.</td>
</tr>
<tr>
<td><strong>PoolingOptions</strong></td>
<td>Specifies the behaviour of connection pool.</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>Serves to supply the server name for login.</td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Used to supply a user name for login.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td><strong>Commit</strong></td>
<td>Commits current transaction.</td>
</tr>
<tr>
<td><strong>Connect</strong></td>
<td>Establishes a connection to the server.</td>
</tr>
<tr>
<td><strong>CreateSQL</strong></td>
<td>Creates a component for queries execution.</td>
</tr>
<tr>
<td><strong>Disconnect</strong></td>
<td>Performs disconnect.</td>
</tr>
<tr>
<td><strong>ExecProc</strong></td>
<td>Allows to execute stored procedure or function providing its name and parameters.</td>
</tr>
<tr>
<td><strong>ExecProcEx</strong></td>
<td>Allows to execute a stored procedure or function.</td>
</tr>
<tr>
<td><strong>ExecSQL</strong></td>
<td>Executes a SQL statement with parameters.</td>
</tr>
</tbody>
</table>
ExecSQLEx | Executes any SQL statement outside the TQuery or TSQL components.
---|---
GetDatabaseNames | Returns a database list from the server.
GetKeyFieldNames | Provides a list of available key field names.
GetStoredProcNames | Returns a list of stored procedures from the server.
GetTableNames | Provides a list of available tables names.
MonitorMessage | Sends a specified message through the TCustomDASQLMonitor component.
Ping | Used to check state of connection to the server.
RemoveFromPool | Marks the connection that should not be returned to the pool after disconnect.
Rollback | Discards all current data changes and ends transaction.
StartTransaction | Begins a new user transaction.

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnectionLost</td>
<td>This event occurs when connection was lost.</td>
</tr>
<tr>
<td>OnError</td>
<td>This event occurs when an error has arisen in the connection.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
**TCustomDAConnection Members** topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectDialog</td>
<td>Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td>ConnectString</td>
<td>Used to specify the connection information, such as: UserName, Password, Server, etc.</td>
</tr>
<tr>
<td>ConvertEOL</td>
<td>Allows customizing line breaks in string fields and parameters.</td>
</tr>
<tr>
<td>InTransaction</td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td>LoginPrompt</td>
<td>Specifies whether a login dialog appears immediately before opening a new connection.</td>
</tr>
<tr>
<td>Options</td>
<td>Specifies the connection behavior.</td>
</tr>
<tr>
<td>Password</td>
<td>Serves to supply a password for login.</td>
</tr>
<tr>
<td>Pooling</td>
<td>Enables or disables using connection pool.</td>
</tr>
<tr>
<td>PoolingOptions</td>
<td>Specifies the behaviour of connection pool.</td>
</tr>
<tr>
<td>Server</td>
<td>Serves to supply the server name for login.</td>
</tr>
<tr>
<td>Username</td>
<td>Used to supply a user name for login.</td>
</tr>
</tbody>
</table>

See Also
- TCustomDAConnection Class
- TCustomDAConnection Class Members
5.11.1.4.2.1 ConnectDialog Property

Allows to link a TCustomConnectDialog component.

Class

TCustomDAConnection

Syntax

property ConnectDialog: TCustomConnectDialog;

Remarks

Use the ConnectDialog property to assign to connection a TCustomConnectDialog component.

See Also

• TCustomConnectDialog

5.11.1.4.2.2 ConnectString Property

Used to specify the connection information, such as: UserName, Password, Server, etc.

Class

TCustomDAConnection

Syntax

property ConnectString: string stored False;

Remarks

PgDAC recognizes an ODBC-like syntax in provider string property values. Within the string, elements are delimited by using a semicolon. Each element consists of a keyword, an equal sign character, and the value passed on initialization. For example:

Server=London1;User ID=nancyd

Connection parameters

The following connection parameters can be used to customize connection:
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoginPrompt</td>
<td>Specifies whether a login dialog appears immediately before opening a new connection.</td>
</tr>
<tr>
<td>Pooling</td>
<td>Enables or disables using connection pool.</td>
</tr>
<tr>
<td>ConnectionLifeTime</td>
<td>Used to specify the maximum time during which an opened connection can be used by connection pool.</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>Used to specify the maximum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td>MinPoolSize</td>
<td>Used to specify the minimum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td>Validate Connection</td>
<td>Used for a connection to be validated when it is returned from the pool.</td>
</tr>
<tr>
<td>Server</td>
<td>Serves to supply the server name for login.</td>
</tr>
<tr>
<td>Username</td>
<td>Used to supply a user name for login.</td>
</tr>
<tr>
<td>Password</td>
<td>Used to supply a user name for login.</td>
</tr>
<tr>
<td>Database</td>
<td>Used to set the name of the database to associate with TPgConnection component.</td>
</tr>
<tr>
<td>Charset</td>
<td>Used to set the character set that PgDAC uses to read and write character data.</td>
</tr>
<tr>
<td>UseUnicode</td>
<td>Used to enable or disable Unicode support.</td>
</tr>
<tr>
<td>Port</td>
<td>Used to specify the port number for the connection.</td>
</tr>
<tr>
<td>ConnectionTimeout</td>
<td>Used to specify the amount of time before an attempt to make a connection is considered unsuccessful.</td>
</tr>
<tr>
<td>ProtocolVersion</td>
<td>Used to set the version of protocol for communication with PostgreSQL server.</td>
</tr>
<tr>
<td>Schema</td>
<td>Used to change the search path of the connection to the specified schema, or get the first value from the search path.</td>
</tr>
<tr>
<td>IPVersion</td>
<td>Used to specify the version of the Internet Protocol.</td>
</tr>
<tr>
<td>P:Devart.Pgdac.TPgConnectionSSLOptions.CACert</td>
<td>Holds the pathname to the certificate authority file.</td>
</tr>
<tr>
<td>P:Devart.Pgdac.TPgConnectionSSLOptions.Cert</td>
<td>Holds the pathname to the certificate file.</td>
</tr>
<tr>
<td>P:Devart.Pgdac.TPgConnectionSSLOptions.CipherList</td>
<td>Holds the list of allowed ciphers to use for SSL encryption.</td>
</tr>
</tbody>
</table>
### ConvertEOL Property

Allows customizing line breaks in string fields and parameters.

**Class**

**TCustomDACConnection**

**Syntax**

```property
ConvertEOL: boolean default False;
```

**Remarks**

Affects the line break behavior in string fields and parameters. When fetching strings (including the TEXT fields) with ConvertEOL = True, dataset converts their line breaks from the LF to CRLF form. And when posting strings to server with ConvertEOL turned on, their line breaks are converted from CRLF to LF form. By default, strings are not converted.

### InTransaction Property

Indicates whether the transaction is active.

**Class**
**TCustomDAConnection**

**Syntax**

```property InTransaction: boolean;```

**Remarks**

Examine the `InTransaction` property at runtime to determine whether user transaction is currently in progress. In other words `InTransaction` is set to True when user explicitly calls `StartTransaction`. Calling `Commit` or `Rollback` sets `InTransaction` to False. The value of the `InTransaction` property cannot be changed directly.

**See Also**
- `StartTransaction`
- `Commit`
- `Rollback`

© 1997-2021 Devart. All Rights Reserved.

5.11.1.4.2.5 LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

**Class**

`TCustomDAConnection`

**Syntax**

```property LoginPrompt default DefValLoginPrompt;```

**Remarks**

Specifies whether a login dialog appears immediately before opening a new connection. If `ConnectDialog` is not specified, the default connect dialog will be shown. The connect dialog will appear only if the PgDacVcl unit appears to the `uses` clause.

© 1997-2021 Devart. All Rights Reserved.

© 2021 Devart
5.11.1.4.2.6 Options Property

Specifies the connection behavior.

Class

TCustomDAConnection

Syntax

property Options: TDAConnectionOptions;

Remarks

Set the properties of Options to specify the behaviour of the connection.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowImplicitConnect</td>
<td>Specifies whether to allow or not implicit connection opening.</td>
</tr>
<tr>
<td>DefaultSortType</td>
<td>Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.</td>
</tr>
<tr>
<td>DisconnectedMode</td>
<td>Used to open a connection only when needed for performing a server call and closes after performing the operation.</td>
</tr>
<tr>
<td>KeepDesignConnected</td>
<td>Used to prevent an application from establishing a connection at the time of startup.</td>
</tr>
<tr>
<td>LocalFailover</td>
<td>If True, the OnConnectionLost event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
</tbody>
</table>

See Also

- Disconnected Mode
- Working in an Unstable Network

© 1997-2021
Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

© 2021 Devart
5.11.1.4.2.7 Password Property

Serves to supply a password for login.

Class

TCustomDAConnection

Syntax

property Password: string stored False;

Remarks

Use the Password property to supply a password to handle server's request for a login.

Warning: Storing hard-coded user name and password entries as property values or in code for the OnLogin event handler can compromise server security.

See Also

- Username
- Server

5.11.1.4.2.8 Pooling Property

Enables or disables using connection pool.

Class

TCustomDAConnection

Syntax

property Pooling: boolean default DefValPooling;

Remarks

Normally, when TCustomDAConnection establishes connection with the server it takes server memory and time resources for allocating new server connection. For example, pooling can be very useful when using disconnect mode. If an application has wide user activity that forces many connect/disconnect operations, it may spend a lot of time on creating connection and sending requests to the server. TCustomDAConnection has
software pool which stores open connections with identical parameters.

Connection pool uses separate thread that validates the pool every 30 seconds. Pool validation consists of checking each connection in the pool. If a connection is broken due to a network problem or another reason, it is deleted from the pool. The validation procedure removes also connections that are not used for a long time even if they are valid from the pool.

Set Pooling to True to enable pooling. Specify correct values for PoolingOptions. Two connections belong to the same pool if they have identical values for the parameters: MinPoolSize, MaxPoolSize, Validate, ConnectionLifeTime

**Note:** Using Pooling := True can cause errors with working with temporary tables.

See Also
- Username
- Password
- PoolingOptions
- Connection Pooling

## 5.11.1.4.2.9 PoolingOptions Property

Specifies the behaviour of connection pool.

### Class

TCustomDACConnection

### Syntax

```
property PoolingOptions: TPoolingOptions;
```

### Remarks

Set the properties of PoolingOptions to specify the behaviour of connection pool.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionLifetime</td>
<td>Used to specify the maximum time during which an opened connection can be used</td>
</tr>
</tbody>
</table>
MaxPoolSize
Used to specify the maximum number of connections that can be opened in connection pool.

MinPoolSize
Used to specify the minimum number of connections that can be opened in the connection pool.

Validate
Used for a connection to be validated when it is returned from the pool.

See Also
- Pooling

Class
TCustomDAConnection

Syntax

```
property Server: string;
```

Remarks
Use the Server property to supply server name to handle server's request for a login.

See Also
- Username
- Password
5.11.1.4.2.11 Username Property

Used to supply a user name for login.

Class
TCustomDAConnection

Syntax

```plaintext
property Username: string;
```

Remarks

Use the Username property to supply a user name to handle server's request for login. If this property is not set, PgDAC tries to connect with the user name.

**Warning**: Storing hard-coded user name and password entries as property values or in code for the OnLogin event handler can compromise server security.

See Also
- Password
- Server

© 1997-2021
Devart. All Rights Reserved.

5.11.1.4.3 Methods

Methods of the TCustomDAConnection class.

For a complete list of the TCustomDAConnection class members, see the TCustomDAConnection Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyUpdates</td>
<td>Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td>Commit</td>
<td>Commits current transaction.</td>
</tr>
<tr>
<td>Connect</td>
<td>Establishes a connection to the server.</td>
</tr>
<tr>
<td>CreateSQL</td>
<td>Creates a component for queries execution.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Performs disconnect.</td>
</tr>
<tr>
<td>ExecProc</td>
<td>Allows to execute stored procedure or function providing its name and parameters.</td>
</tr>
<tr>
<td>ExecProcEx</td>
<td>Allows to execute a stored procedure or function.</td>
</tr>
<tr>
<td>ExecSQL</td>
<td>Executes a SQL statement with parameters.</td>
</tr>
<tr>
<td>ExecSQLEX</td>
<td>Executes any SQL statement outside the TQuery or TSQL components.</td>
</tr>
<tr>
<td>GetDatabaseNames</td>
<td>Returns a database list from the server.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetStoredProcNames</td>
<td>Returns a list of stored procedures from the server.</td>
</tr>
<tr>
<td>GetTableNames</td>
<td>Provides a list of available tables names.</td>
</tr>
<tr>
<td>MonitorMessage</td>
<td>Sends a specified message through the TCustomDASQLMonitor component.</td>
</tr>
<tr>
<td>Ping</td>
<td>Used to check state of connection to the server.</td>
</tr>
<tr>
<td>RemoveFromPool</td>
<td>Marks the connection that should not be returned to the pool after disconnect.</td>
</tr>
<tr>
<td>Rollback</td>
<td>Discards all current data changes and ends transaction.</td>
</tr>
<tr>
<td>StartTransaction</td>
<td>Begins a new user transaction.</td>
</tr>
</tbody>
</table>

See Also
- TCustomDAConnection Class
- TCustomDAConnection Class Members
5.11.4.3.1 ApplyUpdates Method

Applies changes in datasets.

Class

TCustomDAConnection

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyUpdates</td>
<td>Applies changes from all active datasets.</td>
</tr>
<tr>
<td>ApplyUpdates(const DataSets: array of TCustomDADataSet)</td>
<td>Applies changes from the specified datasets.</td>
</tr>
</tbody>
</table>

Applies changes from all active datasets.

Syntax

```delphi
procedure ApplyUpdates; overload; virtual;
```

Remarks

Call the ApplyUpdates method to write all pending cached updates from all active datasets attached to this connection to a database or from specific datasets. The ApplyUpdates method passes cached data to the database for storage, takes care of committing or rolling back transactions, and clearing the cache when the operation is successful.

Using ApplyUpdates for connection is a preferred method of updating datasets rather than calling each individual dataset's ApplyUpdates method.

See Also

- `TMemDataSet.CachedUpdates`
- `TMemDataSet.ApplyUpdates`
Reserved.

Applies changes from the specified datasets.

Class

TCustomDAConnection

Syntax

```pascal
procedure ApplyUpdates(const DataSets: array of TCustomDADataset); overload; virtual;
```

Parameters

DataSets
A list of datasets changes in which are to be applied.

Remarks

Call the ApplyUpdates method to write all pending cached updates from the specified datasets. The ApplyUpdates method passes cached data to the database for storage, takes care of committing or rolling back transactions and clearing the cache when operation is successful.

Using ApplyUpdates for connection is a preferred method of updating datasets rather than calling each individual dataset's ApplyUpdates method.

© 1997-2021 Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback

5.11.1.4.3.2 Commit Method

Commits current transaction.

Class

TCustomDAConnection

Syntax

```pascal
procedure Commit; virtual;
```

Remarks

Call the Commit method to commit current transaction. On commit server writes
permanently all pending data updates associated with the current transaction to the database and then ends the transaction. The current transaction is the last transaction started by calling StartTransaction.

See Also
- Rollback
- StartTransaction
- TCustomPgDataSet.FetchAll

Connect Method

Establishes a connection to the server.

Class
TCustomDAConnection

Syntax

```delphi
procedure Connect; overload:
procedure Connect(const ConnectString: string); overload;
```

Remarks

Call the Connect method to establish a connection to the server. Connect sets the Connected property to True. If LoginPrompt is True, Connect prompts user for login information as required by the server, or otherwise tries to establish a connection using values provided in the Username, Password, and Server properties.

See Also
- Disconnect
- Username
- Password
- Server
- ConnectDialog
5.11.1.4.3.4  CreateSQL Method

Creates a component for queries execution.

Class

`TCustomDAConnection`

Syntax

```plaintext
function CreateSQL: TCustomDASQL; virtual;
```

Return Value

A new instance of the class.

Remarks

Call the CreateSQL to return a new instance of the `TCustomDASQL` class and associates it with this connection object. In the descendant classes this method should be overridden to create an appropriate descendant of the TCustomDASQL component.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.4.3.5  Disconnect Method

Performs disconnect.

Class

`TCustomDAConnection`

Syntax

```plaintext
procedure Disconnect;
```

Remarks

Call the Disconnect method to drop a connection to database. Before the connection component is deactivated, all associated datasets are closed. Calling Disconnect is similar to setting the Connected property to False.

In most cases, closing a connection frees system resources allocated to the connection.

If user transaction is active, e.g. the `InTransaction` flag is set, calling to Disconnect the current user transaction.
Note: If a previously active connection is closed and then reopened, any associated datasets must be individually reopened; reopening the connection does not automatically reopen associated datasets.

See Also
- Connect

5.11.1.4.3.6 ExecProc Method

Allows to execute stored procedure or function providing its name and parameters.

Class
TCustomDAConnection

Syntax

```pascal
function ExecProc(const Name: string; const Params: array of variant): variant; virtual;
```

Parameters

- **Name**
  - Holds the name of the stored procedure or function.

- **Params**
  - Holds the parameters of the stored procedure or function.

Return Value

  - the result of the stored procedure.

Remarks

Allows to execute stored procedure or function providing its name and parameters.

Use the following Name value syntax for executing specific overloaded routine:
"StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded stored procedure, while the second example executes the fifth overloaded procedure.

Assign parameters' values to the Params array in exactly the same order and number as they appear in the stored procedure declaration. Out parameters of the procedure can be accessed with the ParamByName procedure.

If the value of an input parameter was not included to the Params array, parameter default
value is taken. Only parameters at the end of the list can be unincluded to the Params array. If the parameter has no default value, the NULL value is sent.

**Note:** Stored functions unlike stored procedures return result values that are obtained internally through the RESULT parameter. You will no longer have to provide anonymous value in the Params array to describe the result of the function. The stored function result is obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.

For further examples of parameter usage see [ExecSQL](#), [ExecSQLEx](#).

**Example**

For example, having stored function declaration presented in Example 1), you may execute it and retrieve its result with commands presented in Example 2):

```sql
Example 1)
CREATE procedure MY_SUM (      A INTEGER,      B INTEGER)RETURNS (      RESULT INTEGER)
as
begin
  Result = a + b;
end;
Example 2)
Label2.Caption:= MyPgConnection1.ParamByName('Result').AsString;
```

**See Also**
- ExecProcEx
- ExecSQL
- ExecSQLEx

© 1997-2021 Devart. All Rights Reserved.

5.11.1.4.3.7 ExecProcEx Method

Allows to execute a stored procedure or function.

**Class**

TCustomDAConnection
Syntax

```pascal
function ExecProcEx(const Name: string; const Params: array of variant): variant;
```

**Parameters**

*Name*
Holds the stored procedure name.

*Params*
Holds an array of pairs of parameters' names and values.

**Return Value**
the result of the stored procedure.

**Remarks**

Allows to execute a stored procedure or function. Provide the stored procedure name and its parameters to the call of ExecProcEx.

Use the following Name value syntax for executing specific overloaded routine: "StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded stored procedure, while the second example executes the fifth overloaded procedure.

Assign pairs of parameters' names and values to a Params array so that every name comes before its corresponding value when an array is being indexed.

Out parameters of the procedure can be accessed with the ParamByName procedure. If the value for an input parameter was not included to the Params array, the parameter default value is taken. If the parameter has no default value, the NULL value is sent.

**Note:** Stored functions unlike stored procedures return result values that are obtained internally through the RESULT parameter. You will no longer have to provide anonymous value in the Params array to describe the result of the function. Stored function result is obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.

For an example of parameters usage see ExecSQLEx.

**Example**

If you have some stored procedure accepting four parameters, and you want to provide values only for the first and fourth parameters, you should call ExecProcEx in the following way:

```pascal
Connection.ExecProcEx('Some_Stored_Procedure', ['Param_Name1', 'Param_Value1', 'Param_Name4', 'Param_Value4']);
```
5.11.1.4.3.8 ExecSQL Method

Executes a SQL statement with parameters.

Class

TCustomDAConnection

Syntax

```pascal
function ExecSQL(const Text: string): variant;
overload; function ExecSQL(const Text: string; const Params: array of variant): variant; overload; virtual;
```

Parameters

- **Text**
  a SQL statement to be executed.

- **Params**
  Array of parameter values arranged in the same order as they appear in SQL statement.

Return Value

Out parameter with the name Result will hold the result of function having data type dtString. Otherwise returns Null.

Remarks

Use the ExecSQL method to execute any SQL statement outside the TCustomDADataset or TCustomDASQl components. Supply the Params array with the values of parameters arranged in the same order as they appear in a SQL statement which itself is passed to the Text string parameter.

See Also

- ExecSQL
- ExecSQLEx
- ExecProc

© 1997-2021 Devart. All Rights Reserved.
Executes any SQL statement outside the TQuery or TSQL components.

Class

TCustomDAConnection

Syntax

function ExecSQLEx(const Text: string; const Params: array of variant): variant; virtual;

Parameters

Text
a SQL statement to be executed.

Params
Array of parameter values arranged in the same order as they appear in SQL statement.

Return Value
Out parameter with the name Result will hold the result of a function having data type dtString. Otherwise returns Null.

Remarks

Call the ExecSQLEx method to execute any SQL statement outside the TQuery or TSQL components. Supply the Params array with values arranged in pairs of parameter name and its value. This way each parameter name in the array is found on even index values whereas parameter value is on odd index value but right after its parameter name. The parameter pairs must be arranged according to their occurrence in a SQL statement which itself is passed in the Text string parameter.

The Params array must contain all IN and OUT parameters defined in the SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the ExecSQLEx method.

Out parameter with the name Result will hold the result of a function having data type dtString. If neither of the parameters in the Text statement is named Result, ExecSQLEx will return Null.

To get the values of OUT parameters use the ParamByName function.

Example
PgConnection.ExecSQLEx('begin :A:= :B + :C; end;', 
['A', 0, 'B', 5, 'C', 3]);
A:= PgConnection.ParamByName('A').AsInteger;

See Also
- **ExecSQL**

5.11.1.4.3.10  GetDatabaseNames Method

Returns a database list from the server.

Class

**TCustomDAConnection**

Syntax

```pascal
procedure GetDatabaseNames(List: TStrings); virtual;
```

Parameters

- **List**

  A TStrings descendant that will be filled with database names.

Remarks

Populates a string list with the names of databases.

**Note:** Any contents already in the target string list object are eliminated and overwritten by data produced by GetDatabaseNames.

See Also

- **GetTableNames**
- **GetStoredProcNames**

5.11.1.4.3.11  GetKeyFieldNames Method

Provides a list of available key field names.

Class
**TCustomDAConnection**

**Syntax**

```plaintext
procedure GetKeyFieldNames(const TableName: string; List: TStrings); virtual;
```

**Parameters**

- **TableName**
  - Holds the table name

- **List**
  - The list of available key field names

**Return Value**

- Key field name

**Remarks**

Call the GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

**See Also**

- [GetTableNames](#)
- [GetStoredProcNames](#)

---

**5.11.1.4.3.12 GetStoredProcNames Method**

Returns a list of stored procedures from the server.

**Class**

**TCustomDAConnection**

**Syntax**

```plaintext
procedure GetStoredProcNames(List: TStrings; AllProcs: boolean = False); virtual;
```

**Parameters**

- **List**
  - A TStrings descendant that will be filled with the names of stored procedures in the database.
AllProcs

True, if stored procedures from all schemas or including system procedures (depending on the server) are returned. False otherwise.

Remarks

Call the GetStoredProcNames method to get the names of available stored procedures and functions. GetStoredProcNames populates a string list with the names of stored procs in the database. If AllProcs = True, the procedure returns to the List parameter the names of the stored procedures that belong to all schemas; otherwise, List will contain the names of functions that belong to the current schema.

Note: Any contents already in the target string list object are eliminated and overwritten by data produced by GetStoredProcNames.

See Also

- GetDatabaseNames
- GetTableNames

See Also

- GetDatabaseNames
- GetTableNames

© 1997-2021
Devart. All Rights Reserved.

5.11.1.4.3.13 GetTableNames Method

Provides a list of available tables names.

Class

TCustomDAConnection

Syntax

procedure GetTableNames(List: TStrings; AllTables: boolean = False; OnlyTables: boolean = False); virtual;

Parameters

List

A TStrings descendant that will be filled with table names.

AllTables

True, if procedure returns all table names including the names of system tables to the List parameter.

OnlyTables

Remarks
Call the GetTableNames method to get the names of available tables. Populates a string list with the names of tables in the database. If AllTables = True, procedure returns all table names including the names of system tables to the List parameter, otherwise List will not contain the names of system tables. If AllTables = True, the procedure returns to the List parameter the names of the tables that belong to all schemas; otherwise, List will contain the names of the tables that belong to the current schema.

**Note:** Any contents already in the target string list object are eliminated and overwritten by the data produced by GetTableNames.

See Also
- `GetDatabaseNames`
- `GetStoredProcNames`

5.11.1.4.3.14  MonitorMessage Method

Sends a specified message through the `TCustomDASQLMonitor` component.

**Class**

`TCustomDAConnection`

**Syntax**

```delphi
procedure MonitorMessage(const Msg: string);
```

**Parameters**

*Msg*

Message text that will be sent.

**Remarks**

Call the MonitorMessage method to output specified message via the `TCustomDASQLMonitor` component.

See Also
- `TCustomDASQLMonitor`
5.11.1.4.3.15 Ping Method

Used to check state of connection to the server.

Class

TCustomDAConnection

Syntax

procedure Ping;

Remarks

The method is used for checking server connection state.

5.11.1.4.3.16 RemoveFromPool Method

Marks the connection that should not be returned to the pool after disconnect.

Class

TCustomDAConnection

Syntax

procedure RemoveFromPool;

Remarks

Call the RemoveFromPool method to mark the connection that should be deleted after disconnect instead of returning to the connection pool.

See Also

- Pooling
- PoolingOptions
5.11.1.4.3.17 Rollback Method

Discards all current data changes and ends transaction.

Class

TCustomDAConnection

Syntax

```delphi
procedure Rollback; virtual;
```

Remarks

Call the Rollback method to discard all updates, insertions, and deletions of data associated with the current transaction to the database server and then end the transaction. The current transaction is the last transaction started by calling `StartTransaction`.

See Also

- Commit
- StartTransaction
- TCustomPgDataSet.FetchAll

5.11.1.4.3.18 StartTransaction Method

 Begins a new user transaction.

Class

TCustomDAConnection

Syntax

```delphi
procedure StartTransaction; virtual;
```

Remarks

Call the StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the `InTransaction` property. If InTransaction is True, indicating that a transaction is already in progress, a subsequent call to StartTransaction without first calling `Commit` or `Rollback` to
end the current transaction raises EDatabaseError. Calling StartTransaction when connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes, or Rollback to cancel them.

See Also
- Commit
- Rollback
- InTransaction

5.11.1.4.4 Events

Events of the TCustomDAConnection class.

For a complete list of the TCustomDAConnection class members, see the TCustomDAConnection Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnectionLost</td>
<td>This event occurs when connection was lost.</td>
</tr>
<tr>
<td>OnError</td>
<td>This event occurs when an error has arisen in the connection.</td>
</tr>
</tbody>
</table>

See Also
- TCustomDAConnection Class
- TCustomDAConnection Class Members

© 1997-2021
Devart. All Rights Reserved.  Request Support  DAC Forum  Provide Feedback
**TCustomDAConnection**

**Syntax**

```
property OnConnectionLost: TConnectionLostEvent;
```

**Remarks**

Write the OnConnectionLost event handler to process fatal errors and perform failover.

**Note:** To use the OnConnectionLost event handler, you should explicitly add the MemData unit to the 'uses' list and set the TCustomDAConnection.Options.LocalFailover property to True.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.4.2 OnError Event

This event occurs when an error has arisen in the connection.

**Class**

**TCustomDAConnection**

**Syntax**

```
property OnError: TDAConnectionErrorEvent;
```

**Remarks**

Write the OnError event handler to respond to errors that arise with connection. Check the E parameter to get the error code. Set the Fail parameter to False to prevent an error dialog from being displayed and to raise the EAbort exception to cancel current operation. The default value of Fail is True.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5 TCustomDADataSet Class

Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.

For a list of all members of this type, see **TCustomDADataSet** members.
Unit

**DBAccess**

Syntax

```pascal
TCustomDADataSet = class(TMemDataSet);
```

Remarks

TCustomDADataSet encapsulates general set of properties, events, and methods for working with data accessed through various database engines. All database-specific features are supported by descendants of TCustomDADataSet.

Applications should not use TCustomDADataSet objects directly.

Inheritance Hierarchy

- `TMemDataSet`
- `TCustomDADataSet`

© 1997-2021 Devart. All Rights Reserved.

**5.11.1.5.1 Members**

`TCustomDADataSet` class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BaseSQL</strong></td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong></td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td>Used to set data type</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>DetailFields</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>FetchRows</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>(inherited from TMemDataSet) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>IsQuery</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from TMemDataSet) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>(inherited from TMemDataSet) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MacroCount</strong></td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>Macros</strong></td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong></td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>MasterSource</strong></td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to specify the behaviour of TCustomDADataSet object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong></td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>Used to indicate the number</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong></td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the <code>TCustomDADataSet.RefreshRecord</code> procedure.</td>
</tr>
<tr>
<td><strong>SQLUpdate</strong></td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td><strong>UniDirectional</strong></td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

Methods
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from <strong>TMemDataSet</strong>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>BreakExec</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange (inherited from <strong>TMemDataSet</strong>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>CreateBlobStream</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td>DeferredPost (inherited from <strong>TMemDataSet</strong>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>EditRangeEnd (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Fetched</td>
<td>Used to find out whether TCustomDADataSet has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>Used to learn whether TCustomDADataSet is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>Used to learn whether TCustomDADataSet is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td>FindParam</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>GetBlob (inherited from TMemDataSet)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>GetDataType</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>Sets the current record in this dataset similar to the</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Excludes features that don't need to be included to the <strong>TMemDataSet.Locate</strong> method of TDataSet.</td>
</tr>
<tr>
<td><strong>Lock</strong></td>
<td>Locks the current record.</td>
</tr>
<tr>
<td><strong>MacroByName</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td>Sets or uses parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>RefreshRecord</strong></td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong></td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong></td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
</tbody>
</table>
SetRangeEnd (inherited from **TMemDataSet**)  
Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

SetRangeStart (inherited from **TMemDataSet**)  
Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

**SQLSaved**  
Determines if the SQL property value was saved to the BaseSQL property.

**UnLock**  
Releases a record lock.

**UnPrepare** (inherited from **TMemDataSet**)  
Frees the resources allocated for a previously prepared query on the server and client sides.

**UpdateResult** (inherited from **TMemDataSet**)  
Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

**UpdateStatus** (inherited from **TMemDataSet**)  
Indicates the current update status for the dataset when cached updates are enabled.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong></td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td><strong>AfterFetch</strong></td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td><strong>AfterUpdateExecute</strong></td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td><strong>BeforeFetch</strong></td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td><strong>BeforeUpdateExecute</strong></td>
<td>Occurs before executing</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BaseSQL</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>DataTypeMap</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>DetailFields</td>
<td>Used to specify the fields that correspond to the foreign key fields from...</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Disconnected</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>FetchRows</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>(inherited from TMemDataSet) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>IsQuery</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from TMemDataSet) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>(inherited from TMemDataSet) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>(inherited from TMemDataSet) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change</td>
</tr>
</tbody>
</table>

MasterFields when building master/detail relationship.
<table>
<thead>
<tr>
<th><strong>MasterFields</strong></th>
<th>SQL queries easily.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MasterSource</strong></td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to specify the behaviour of TCustomDADataSet object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong></td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td><strong>Prepared</strong> (inherited from TMemDataSet)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong> (inherited from TMemDataSet)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong></td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td><strong>SQLUpdate</strong></td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td><strong>UniDirectional</strong></td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

**See Also**
- **TCustomDADataset Class**
- **TCustomDADataset Class Members**

© 1997-2021 Devart. All Rights Reserved.
5.11.1.5.2.1 BaseSQL Property

Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.

Class

TCustomDADataSet

Syntax

```property BaseSQL: string;```

Remarks

Use the BaseSQL property to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL, only macros are expanded. SQL text with all these changes can be returned by FinalSQL.

See Also

- FinalSQL
- AddWhere
- SaveSQL
- SQLSaved
- RestoreSQL

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.2.2 Conditions Property

Used to add WHERE conditions to a query

Class

TCustomDADataSet

Syntax

```property Conditions: TDAConditions stored False;```

See Also

- TDAConditions
5.11.1.5.2.3 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

TCustomDADataSet

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

5.11.1.5.2.4 DataTypeMap Property

Used to set data type mapping rules

Class

TCustomDADataSet

Syntax

```
property DataTypeMap: TDAMapRules stored IsMapRulesStored;
```

See Also

- TDAMapRules

© 1997-2021

Devar. All Rights Reserved.
5.11.1.5.2.5 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

TCustomDADataSet

Syntax

```pascal
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

**Note:** If TPgSQLMonitor is used in the project and the TPgSQLMonitor.Active property is set to False, the debug window is not displayed.

See Also

- TCustomDASQL.Debug

5.11.1.5.2.6 DetailFields Property

Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.

Class

TCustomDADataSet

Syntax

```pascal
property DetailFields: string;
```
Remarks

Use the DetailFields property to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship. DetailFields is a string containing one or more field names in the detail table. Separate field names with semicolons.

Use Field Link Designer to set the value in design time.

See Also
- MasterFields
- MasterSource

Class

TCustomDADataSet

Syntax

```
property Disconnected: boolean;
```

Remarks

Set the Disconnected property to True to keep dataset opened after connection is closed.

Class

TCustomDADataSet

Syntax

```
property FetchRows: integer default 25;
```

Remarks

Used to define the number of rows to be transferred across the network at the same time.
Remarks

The number of rows that will be transferred across the network at the same time. This property can have a great impact on performance. So it is preferable to choose the optimal value of the FetchRows property for each SQL statement and software/hardware configuration experimentally.

The default value is 25.

Class

TCustomDADataSet

Syntax

property FilterSQL: string;

Remarks

The FilterSQL property is similar to the Filter property, but it changes the WHERE clause of SELECT statement and reopens query. Syntax is the same to the WHERE clause.

Note: the FilterSQL property adds a value to the WHERE condition as is. If you expect this value to be enclosed in brackets, you should bracket it explicitly.

Example

Query1.FilterSQL := 'Dept >= 20 and DName LIKE ''M%'';

See Also

• AddWhere
5.11.1.5.2.10  FinalSQL Property

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Class

TCustomDADataset

Syntax

property FinalSQL: string;

Remarks

Use FinalSQL to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros. This is the exact statement that will be passed on to the database server.

See Also

- FinalSQL
- AddWhere
- SaveSQL
- SQLSaved
- RestoreSQL
- BaseSQL

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.2.11  IsQuery Property

Used to check whether SQL statement returns rows.

Class

TCustomDADataset

Syntax

property IsQuery: boolean;

Remarks
After the TCustomDADataSet component is prepared, the IsQuery property returns True if SQL statement is a SELECT query.

Use the IsQuery property to check whether the SQL statement returns rows or not.

IsQuery is a read-only property. Reading IsQuery on unprepared dataset raises an exception.

5.11.1.5.2.12 KeyFields Property

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.

Class

TCustomDADataSet

Syntax

property KeyFields: string;

Remarks

TCustomDADataSet uses the KeyFields property to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database. For this feature KeyFields may hold a list of semicolon-delimited field names. If KeyFields is not defined before opening a dataset, TCustomDADataSet requests information about primary keys by sending an additional query to the server.

See Also

- SQLDelete
- SQLInsert
- SQLRefresh
- SQLUpdate

© 1997-2021 Devart. All Rights Reserved.
5.11.1.5.2.13  MacroCount Property

Used to get the number of macros associated with the Macros property.

Class

TCustomDADataSet

Syntax

```
property MacroCount: word;
```

Remarks

Use the MacroCount property to get the number of macros associated with the Macros property.

See Also

- Macros

© 1997-2021
Devart. All Rights Reserved.

5.11.1.5.2.14  Macros Property

Makes it possible to change SQL queries easily.

Class

TCustomDADataSet

Syntax

```
property Macros: TMacros stored False;
```

Remarks

With the help of macros you can easily change SQL query text at design- or runtime. Marcos extend abilities of parameters and allow to change conditions in a WHERE clause or sort order in an ORDER BY clause. You just insert &MacroName in the SQL query text and change value of macro in the Macro property editor at design time or call the MacroByName function at run time. At the time of opening the query macro is replaced by its value.

Example
Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

Class

TCustomDADataset

Syntax

```property`` MasterFields: ``string``;

Remarks

Use the MasterFields property after setting the MasterSource property to specify the names of one or more fields that are used as foreign keys for this dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in these fields are used to select corresponding records in this table for display.

Use Field Link Designer to set the values at design time after setting the MasterSource property.

See Also

- DetailFields
- MasterSource
5.11.1.5.2.16  MasterSource Property

Used to specify the data source component which binds current dataset to the master one.

Class

TCustomDADataset

Syntax

```
property MasterSource: TDataSource;
```

Remarks

The MasterSource property specifies the data source component which binds current dataset to the master one.

TCustomDADataset uses MasterSource to extract foreign key fields values from the master dataset when building master/detail relationship between two datasets. MasterSource must point to another dataset; it cannot point to this dataset component.

When MasterSource is not `nil` dataset fills parameter values with corresponding field values from the current record of the master dataset.

**Note:** Do not set the DataSource property when building master/detail relationships. Although it points to the same object as the MasterSource property, it may lead to undesirable results.

See Also

- **MasterFields**
- **DetailFields**
- **Master/Detail Relationships**
5.11.1.5.2.17 Options Property

Used to specify the behaviour of TCustomDADataSet object.

Class

TCustomDADataSet

Syntax

```
property Options: TDDataSetOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TCustomDADataSet object.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPrepare</td>
<td>Used to execute automatic Prepare on the query execution.</td>
</tr>
<tr>
<td>CacheCalcFields</td>
<td>Used to enable caching of the TField.Calculated and TField.Lookup fields.</td>
</tr>
<tr>
<td>CompressBlobMode</td>
<td>Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td>DefaultValues</td>
<td>Used to request default values/expressions from the server and assign them to the DefaultExpression property.</td>
</tr>
<tr>
<td>DetailDelay</td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td>FieldsOrigin</td>
<td>Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.</td>
</tr>
<tr>
<td>FlatBuffers</td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td>InsertAllSetFields</td>
<td>Used to include all set dataset fields in the generated INSERT statement</td>
</tr>
<tr>
<td>LocalMasterDetail</td>
<td>Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.</td>
</tr>
<tr>
<td>LongStrings</td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td><strong>MasterFieldsNullable</strong></td>
<td>Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>NumberRange</strong></td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloaField to appropriate values.</td>
</tr>
<tr>
<td><strong>QueryRecCount</strong></td>
<td>Used for TCustomDADataset to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td><strong>QuoteNames</strong></td>
<td>Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td><strong>RemoveOnRefresh</strong></td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td><strong>RequiredFields</strong></td>
<td>Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td><strong>ReturnParams</strong></td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td><strong>SetFieldsReadOnly</strong></td>
<td>Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.</td>
</tr>
<tr>
<td><strong>StrictUpdate</strong></td>
<td>Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.</td>
</tr>
<tr>
<td><strong>TrimFixedChar</strong></td>
<td>Specifies whether to discard all trailing spaces in the string fields of a dataset.</td>
</tr>
<tr>
<td><strong>UpdateAllFields</strong></td>
<td>Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
<tr>
<td><strong>UpdateBatchSize</strong></td>
<td>Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.</td>
</tr>
</tbody>
</table>

See Also
- **Master/Detail Relationships**
- **TMemDataSet.CachedUpdates**

© 1997-2021 [Request Support]  [DAC Forum]  [Provide Feedback]
5.11.1.5.2.18  ParamCheck Property

Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

Class

TCustomDADataSet

Syntax

```plaintext
property ParamCheck: boolean default True;
```

Remarks

Use the ParamCheck property to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

Set ParamCheck to True to let dataset automatically generate the Params property for the dataset based on a SQL statement.

Setting ParamCheck to False can be used if the dataset component passes to a server the DDL statements that contain, for example, declarations of stored procedures which themselves will accept parameterized values. The default value is True.

See Also

• Params

5.11.1.5.2.19  ParamCount Property

Used to indicate how many parameters are there in the Params property.

Class

TCustomDADataSet

Syntax

```plaintext
property ParamCount: word;
```
Remarks
Use the ParamCount property to determine how many parameters are there in the Params property.

See Also
• Params

5.11.1.5.2.20  Params Property

Used to view and set parameter names, values, and data types dynamically.

Class
TCustomDADataSet

Syntax

```pascal
property Params: TDAParams stored False;
```

Remarks
Contains the parameters for a query's SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information). Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

See Also
• ParamByName
• Macros

© 1997-2021
Devart. All Rights Reserved.

Request Support   DAC Forum   Provide Feedback
5.11.1.5.2.21 ReadOnly Property

Used to prevent users from updating, inserting, or deleting data in the dataset.

Class

TCustomDADataSet

Syntax

property ReadOnly: boolean default False;

Remarks

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is False, meaning that users can potentially alter data stored in the dataset.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to True.

When ReadOnly is True, the dataset's CanModify property is False.

5.11.1.5.2.22 RefreshOptions Property

Used to indicate when the editing record is refreshed.

Class

TCustomDADataSet

Syntax

property RefreshOptions: TRefreshOptions default [];

Remarks

Use the RefreshOptions property to determine when the editing record is refreshed.

Refresh is performed by the RefreshRecord method.

It queries the current record and replaces one in the dataset. Refresh record is useful when the table has triggers or the table fields have default values. Use roBeforeEdit to get actual data before editing.
The default value is []).

See Also
• RefreshRecord

5.11.1.5.2.23  RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class
TCustomDADataSet

Syntax

property RowsAffected: integer;

Remarks
Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

5.11.1.5.2.24  SQL Property

Used to provide a SQL statement that a query component executes when its Open method is called.

Class
TCustomDADataSet

Syntax

property SQL: TStrings;

Remarks
Use the SQL property to provide a SQL statement that a query component executes when its Open method is called. At the design time the SQL property can be edited by invoking the String List editor in Object Inspector.

When SQL is changed, TCustomDADataSet calls Close and UnPrepare.

See Also
- SQLInsert
- SQLUpdate
- SQLDelete
- SQLRefresh

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.2.25 SQLDelete Property

Used to specify a SQL statement that will be used when applying a deletion to a record.

Class
TCustomDADataSet

Syntax

property SQLDelete: TStrings;

Remarks
Use the SQLDelete property to specify the SQL statement that will be used when applying a deletion to a record. Statements can be parameterized queries.

To create a SQLDelete statement at design-time, use the query statements editor.

Example

```
DELETE FROM Orders
WHERE OrderID = :Old_OrderID
```

See Also
- SQL
- SQLInsert
- SQLUpdate
5.11.1.5.2.26  SQLInsert Property

Used to specify the SQL statement that will be used when applying an insertion to a dataset.

Class

TCustomDADataSet

Syntax

```plaintext
property SQLInsert: TStrings;
```

Remarks

Use the SQLInsert property to specify the SQL statement that will be used when applying an insertion to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. Parameters prefixed with OLD_ allow using current values of fields prior to the actual operation.

Use ReturnParam to return OUT parameters back to dataset.

To create a SQLInsert statement at design-time, use the query statements editor.

See Also

- SQL
- SQLUpdate
- SQLDelete
- SQLRefresh

5.11.1.5.2.27  SQLLock Property

Used to specify a SQL statement that will be used to perform a record lock.

Class

TCustomDADataSet
Syntax

```delphi
property SQLLock: TStrings;
```

Remarks

Use the SQLLock property to specify a SQL statement that will be used to perform a record lock. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

To create a SQLLock statement at design-time, the use query statement editor.

See Also

- SQL
- SQLInsert
- SQLUpdate
- SQLDelete
- SQLRefresh

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.2.28 SQLRecCount Property

Used to specify the SQL statement that is used to get the record count when opening a dataset.

Class

`TCustomDADataSet`

Syntax

```delphi
property SQLRecCount: TStrings;
```

Remarks

Use the SQLRecCount property to specify the SQL statement that is used to get the record count when opening a dataset. The SQL statement is used if the `TDADatasetOptions.QueryRecCount` property is True, and the `TCustomDADataSet.FetchAll` property is False. Is not used if the FetchAll property is True.
To create a SQLRecCount statement at design-time, use the query statements editor.

See Also
- SQLInsert
- SQLUpdate
- SQLDelete
- SQLRefresh
- TDADatasetOptions
- FetchingAll

5.11.1.5.2.29 SQLRefresh Property

Used to specify a SQL statement that will be used to refresh current record by calling the RefreshRecord procedure.

Class
TCustomDADataset

Syntax

```property```
SQLRefresh: TStrings;
```

Remarks

Use the SQLRefresh property to specify a SQL statement that will be used to refresh current record by calling the RefreshRecord procedure.

Different behavior is observed when the SQLRefresh property is assigned with a single WHERE clause that holds frequently altered search condition. In this case the WHERE clause from SQLRefresh is combined with the same clause of the SELECT statement in a SQL property and this final query is then sent to the database server.

To create a SQLRefresh statement at design-time, use the query statements editor.

Example

```sql```
SELECT Shipname FROM Orders
WHERE
    OrderID = :OrderID
```

© 2021 Devart
5.11.1.5.2.30  SQLUpdate Property

Used to specify a SQL statement that will be used when applying an update to a dataset.

Class

TCustomDADataset

Syntax

```pascal
property SQLUpdate: TStrings;
```

Remarks

Use the SQLUpdate property to specify a SQL statement that will be used when applying an update to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

Use ReturnParam to return OUT parameters back to the dataset.

To create a SQLUpdate statement at design-time, use the query statement editor.

Example

```sql
UPDATE Orders
  set
    ShipName = :ShipName
WHERE
  OrderID = :Old_OrderID
```

See Also

- SQL
- SQLInsert
5.11.1.5.2.31 UniDirectional Property

Used if an application does not need bidirectional access to records in the result set.

Class

TCustomDADataset

Syntax

property UniDirectional: boolean default False;

Remarks

Traditionally SQL cursors are unidirectional. They can travel only forward through a dataset. TCustomDADataset, however, permits bidirectional travelling by caching records. If an application does not need bidirectional access to the records in the result set, set UniDirectional to True. When UniDirectional is True, an application requires less memory and performance is improved. However, UniDirectional datasets cannot be modified. In FetchAll=False mode data is fetched on demand. When UniDirectional is set to True, data is fetched on demand as well, but obtained rows are not cached except for the current row. In case if the Unidirectional property is True, the FetchAll property will be automatically set to False. And if the FetchAll property is True, the Unidirectional property will be automatically set to False. The default value of UniDirectional is False, enabling forward and backward navigation.

Note: Pay attention to the specificity of using the FetchAll property=False

See Also

• TPgQuery.FetchAll
5.11.1.5.3 Methods

Methods of the `TCustomDADataSet` class.

For a complete list of the `TCustomDADataSet` class members, see the [TCustomDADataSet Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td><strong>ApplyRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>BreakExec</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange (inherited from <code>TMemDataSet</code>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates (inherited from <code>TMemDataSet</code>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates (inherited from <code>TMemDataSet</code>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>CreateBlobStream</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td>DeferredPost (inherited from <code>TMemDataSet</code>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>EditRangeEnd (inherited from <code>TMemDataSet</code>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Fetched</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td>FindParam</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>GetBlob (inherited from TMemDataSet)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>GetDataType</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td>Locate (inherited from TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx (inherited from TMemDataSet)</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Lock</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Sets or uses parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td>RestoreUpdates (inherited from TMemDataSet)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord (inherited from TMemDataSet)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveSQL</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SQLSaved</strong></td>
<td>Determines if the <strong>SQL</strong> property value was saved to the <strong>BaseSQL</strong> property.</td>
</tr>
<tr>
<td><strong>UnLock</strong></td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdateStatus</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

See Also

- **TCustomDADataSet Class**
- **TCustomDADataSet Class Members**
5.11.1.5.3.1  AddWhere Method

Adds condition to the WHERE clause of SELECT statement in the SQL property.

Class

TCustomDADataSet

Syntax

```pascal
procedure AddWhere(const Condition: string);
```

Parameters

- **Condition**
  
  Holds the condition that will be added to the WHERE clause.

Remarks

Call the AddWhere method to add a condition to the WHERE clause of SELECT statement in the SQL property.

If SELECT has no WHERE clause, AddWhere creates it.

**Note:** the AddWhere method is implicitly called by RefreshRecord. The AddWhere method works for the SELECT statements only.

**Note:** the AddWhere method adds a value to the WHERE condition as is. If you expect this value to be enclosed in brackets, you should bracket it explicitly.

See Also

- DeleteWhere

© 1997-2021 Devart. All Rights Reserved.
procedure BreakExec; virtual;

Remarks
Call the BreakExec method to break execution of the SQL statement on the server. It makes sense to only call BreakExec from another thread.

See Also
- TCustomDADataSet.Execute
- TCustomDASQL.BreakExec

5.11.1.5.3.3 CreateBlobStream Method

Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.

Class
TCustomDADataSet

Syntax
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override;

Parameters
Field
Holds the BLOB field for reading data from or writing data to from a stream.

Mode
Holds the stream mode, for which the stream will be used.

Return Value
The BLOB Stream.

Remarks
Call the CreateBlobStream method to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter. It must be a TBlobField component. You can specify whether the stream will be used for reading, writing, or updating the contents of the field with the Mode parameter.
5.11.1.5.3.4 DeleteWhere Method

Removes WHERE clause from the SQL property and assigns the BaseSQL property.

Class

TCustomDADataset

Syntax

procedure DeleteWhere;

Remarks

Call the DeleteWhere method to remove WHERE clause from the SQL property and assign BaseSQL.

See Also

- AddWhere
- BaseSQL

© 1997-2021
Devart. All Rights Reserved.

5.11.1.5.3.5 Execute Method

Executes a SQL statement on the server.

Class

TCustomDADataset

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Execute(Iter: integer; Offset: integer)</td>
<td>Used to perform Batch operations.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.
Executes a SQL statement on the server.

Class
TCustomDADataset

Syntax
```
procedure Execute; overload; virtual;
```

Remarks
Call the Execute method to execute an SQL statement on the server. If SQL statement is a SELECT query, Execute calls the Open method.

Execute implicitly prepares SQL statement by calling the TCustomDADataset.Prepare method if the TCustomDADataset.Options option is set to True and the statement has not been prepared yet. To speed up the performance in case of multiple Execute calls, an application should call Prepare before calling the Execute method for the first time.

See Also
- TCustomDADataset.AfterExecute
- TCustomDADataset.Executing
- TCustomDADataset.Prepare

© 1997-2021 Devart. All Rights Reserved.

Used to perform Batch operations.

Class
TCustomDADataset

Syntax
```
procedure Execute(Iter: integer; Offset: integer = 0); overload; virtual;
```

Parameters
- **Iter**: Specifies the number of inserted rows.
- **Offset**:
Points the array element, which the Batch operation starts from. 0 by default.

Remarks
The Execute method executes the specified batch SQL query. See the [Batch operations](https://www.devart.com/dotnet/docs/batch-operations) article for samples.

See Also
- [Batch operations](https://www.devart.com/dotnet/docs/batch-operations)

5.11.1.5.3.6 Executing Method

Indicates whether SQL statement is still being executed.

**Class**

`TCustomDADataSet`

**Syntax**

```delphi
function Executing: boolean;
```

**Return Value**

True, if SQL statement is still being executed.

Remarks
Check Executing to learn whether TCustomDADataSet is still executing SQL statement.

5.11.1.5.3.7 Fetched Method

Used to find out whether TCustomDADataSet has fetched all rows.

**Class**

`TCustomDADataSet`

**Syntax**

```delphi
function Fetched: boolean; virtual;
```
Return Value

True, if all rows have been fetched.

Remarks

Call the Fetched method to find out whether TCustomDADataSet has fetched all rows.

See Also

• Fetching

Syntax

```delphi
function Fetching: boolean;
```

Return Value

True, if TCustomDADataSet is still fetching rows.

Remarks

Check Fetching to learn whether TCustomDADataSet is still fetching rows. Use the Fetching method if NonBlocking is True.

See Also

• Executing

Used to learn whether TCustomDADataSet is fetching all rows to the end.

Class

TCustomDADataSet
**TCustomDADataSet**

**Syntax**

```delphi
function FetchingAll: boolean;
```

**Return Value**

True, if TCustomDADataSet is fetching all rows to the end.

**Remarks**

Check FetchingAll to learn whether TCustomDADataSet is fetching all rows to the end.

**See Also**

- Executing

© 1997-2021
Devart. All Rights Reserved.

---

5.11.1.5.3.10 FindKey Method

Searches for a record which contains specified field values.

**Class**

TCustomDADataSet

**Syntax**

```delphi
function FindKey(const KeyValues: array of System.TVarRec): Boolean;
```

**Parameters**

- **KeyValues**
  
  Holds a key.

**Remarks**

Call the FindKey method to search for a specific record in a dataset. KeyValues holds a comma-delimited array of field values, that is called a key.

This function is provided for BDE compatibility only. It is recommended to use functions **TMemDataSet.Locate** and **TMemDataSet.LocateEx** for the record search.
Reserve.

5.11.1.5.3.11 FindMacro Method

Finds a macro with the specified name.

Class

TCustomDADataset

Syntax

function FindMacro(const Value: string): TMacro;

Parameters

Value

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

See Also

- TMacro
- Macros
- MacroByName

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.3.12 FindNearest Method

Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.

Class

TCustomDADataset

Syntax
**procedure** FindNearest(const KeyValues: array of System.TVarRec);

### Parameters

**KeyValues**
Holds the values of the record key fields to which the cursor should be moved.

### Remarks

Call the FindNearest method to move the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter. If there are no records that match or exceed the specified criteria, the cursor will not move.

This function is provided for BDE compatibility only. It is recommended to use functions `TMemDataSet.Locate` and `TMemDataSet.LocateEx` for the record search.

### See Also
- `TMemDataSet.Locate`
- `TMemDataSet.LocateEx`
- `FindKey`

---

**function** FindParam(const Value: string): TDAParam;

### Parameters

**Value**
Holds the name of the param for which to search.

### Return Value
the TDAParam object for the specified Name. Otherwise it returns nil.

### Remarks

Call the FindParam method to determine if a specified param component exists in a dataset.
Name is the name of the param for which to search. If FindParam finds a param with a matching name, it returns a TDAParam object for the specified Name. Otherwise it returns nil.

See Also
- Params
- ParamByName

Parameters
- FieldName
  Holds the name of the field.

Return Value
- internal field types defined in MemData and accompanying modules.

Remarks
Call the GetDataType method to return internal field types defined in the MemData and accompanying modules. Internal field data types extend the TFieldType type of VCL by specific database server data types. For example, ftString, ftFile, ftObject.
**TCustomDADataSet**

Syntax

```delphi
function GetFieldObject(Field: TField): TSharedObject;
overload;
function GetFieldObject(Field: TField; RecBuf: TRecordBuffer): TSharedObject; overload;
function GetFieldObject(FieldDesc: TFieldDesc): TSharedObject;
overload;
function GetFieldObject(FieldDesc: TFieldDesc; RecBuf: TRecordBuffer): TSharedObject; overload;
function GetFieldObject(const FieldName: string): TSharedObject; overload;
```

**Parameters**

*FieldName*

Holds the field name.

**Return Value**

multireference shared object.

**Remarks**

Call the GetFieldObject method to return a multireference shared object from field. If field does not hold one of the TSharedObject descendants, GetFieldObject raises an exception.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.3.16 GetFieldPrecision Method

Retrieves the precision of a number field.

**Class**

*TCustomDADataSet*

**Syntax**

```delphi
function GetFieldPrecision(const FieldName: string): integer;
```

**Parameters**

*FieldName*

Holds the existing field name.

**Return Value**

precision of number field.

**Remarks**
Call the GetFieldPrecision method to retrieve the precision of a number field. FieldName is the name of an existing field.

See Also
• GetFieldScale

5.11.1.5.3.17 GetFieldScale Method

Retrieves the scale of a number field.

Class
TCustomDADataSet

Syntax

function GetFieldScale(const FieldName: string): integer;

Parameters

FieldName
  Holds the existing field name.

Return Value
  the scale of the number field.

Remarks

Call the GetFieldScale method to retrieve the scale of a number field. FieldName is the name of an existing field.

See Also
• GetFieldPrecision

5.11.1.5.3.18 GetKeyFieldNames Method

Provides a list of available key field names.

Class
**TCustomDADataSet**

**Syntax**

```pascal
procedure GetKeyFieldNames(List: TStrings);
```

**Parameters**

- **List**
  - The list of available key field names

**Return Value**

- Key field name

**Remarks**

Call the GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

**See Also**

- `TCustomDAConnection.GetTableNames`
- `TCustomDAConnection.GetStoredProcNames`

---

**GetOrderBy Method**

Retrieves an ORDER BY clause from a SQL statement.

**Class**

`TCustomDADataSet`

**Syntax**

```pascal
function GetOrderBy: string;
```

**Return Value**

- an ORDER BY clause from the SQL statement.

**Remarks**

Call the GetOrderBy method to retrieve an ORDER BY clause from a SQL statement.

**Note:** GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.
See Also
  • SetOrderBy

Class
TCustomDADataSet

Syntax

procedure GotoCurrent(DataSet: TCustomDADataSet);

Parameters

DataSet
  Holds the TCustomDADataSet descendant to synchronize the record position with.

Remarks

Call the GotoCurrent method to set the current record in this dataset similar to the current
record in another dataset. The key fields in both these DataSets must be coincident.

See Also
  • TMemDataSet.Locate
  • TMemDataSet.LocateEx

5.11.1.5.3.21 Lock Method

Locks the current record.

Class
TCustomDADataSet

Syntax
procedure Lock; virtual;

Remarks
Call the Lock method to lock the current record by executing the statement that is defined in the SQLLock property.

The Lock method sets the savepoint with the name LOCK_ + <component_name>.

See Also
• UnLock

MacroByName Method

Finds a macro with the specified name.

Class
TCustomDADataSet

Syntax
function MacroByName(const Value: string): TMacro;

Parameters
Value
Holds the name of a macro to search for.

Return Value
TMacro object if a match is found.

Remarks
Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.
Sets or uses parameter information for a specific parameter based on its name.

Class

TCustomDADataset

Syntax

function ParamByName(const Value: string): TDAParam;

Parameters

Value

Holds the name of the parameter for which to retrieve information.

Return Value

a TDAParam object.

Remarks

Call the ParamByName method to set or use parameter information for a specific parameter based on its name. Name is the name of the parameter for which to retrieve information. ParamByName is used to set a parameter's value at runtime and returns a TDAParam object.

Example

The following statement retrieves the current value of a parameter called "Contact" into an edit box:
Edit1.Text := Query1.ParamsByName('Contact').AsString;

See Also
- Params
- FindParam

5.11.1.5.3.24 Prepare Method

Allocates, opens, and parses cursor for a query.

Class
TCustomDADataSet

Syntax

procedure Prepare; override;

Remarks
Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also
- TMemDataSet.Prepared
- TMemDataSet.UnPrepare
- Options

5.11.1.5.3.25 RefreshRecord Method

Actualizes field values for the current record.

Class
**TCustomDADataSet**

**Syntax**

```pascal
procedure RefreshRecord;
```

**Remarks**

Call the RefreshRecord method to actualize field values for the current record. RefreshRecord performs query to database and refetches new field values from the returned cursor.

**See Also**

- RefreshOptions
- SQLRefresh

© 1997-2021 Devart. All Rights Reserved.

5.11.5.3.26 RestoreSQL Method

Restores the SQL property modified by AddWhere and SetOrderBy.

**Class**

**TCustomDADataSet**

**Syntax**

```pascal
procedure RestoreSQL;
```

**Remarks**

Call the RestoreSQL method to restore the SQL property modified by AddWhere and SetOrderBy.

**See Also**

- AddWhere
- SetOrderBy
- SaveSQL
- SQLSaved

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
5.11.1.5.3.27  SaveSQL Method

Saves the SQL property value to BaseSQL.

Class

TCustomDADataSet

Syntax

procedure SaveSQL;

Remarks

Call the SaveSQL method to save the SQL property value to the BaseSQL property.

See Also

- SQLSaved
- RestoreSQL
- BaseSQL

5.11.1.5.3.28  SetOrderBy Method

Builds an ORDER BY clause of a SELECT statement.

Class

TCustomDADataSet

Syntax

procedure SetOrderBy(const Fields: string);

Parameters

Fields

Holds the names of the fields which will be added to the ORDER BY clause.

Remarks

Call the SetOrderBy method to build an ORDER BY clause of a SELECT statement. The
fields are identified by the comma-delimited field names.

**Note:** The GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.

### Example

```sql
Query1.SetOrderBy('DeptNo;DName');
```

### See Also

- `GetOrderBy`

### 5.11.1.5.3.29 SQLSaved Method

Determines if the `SQL` property value was saved to the `BaseSQL` property.

**Class**

`TCustomDADataSet`

**Syntax**

```pascal
function SQLSaved: boolean;
```

**Return Value**

True, if the SQL property value was saved to the BaseSQL property.

**Remarks**

Call the SQLSaved method to know whether the `SQL` property value was saved to the `BaseSQL` property.

### 5.11.1.5.3.30 UnLock Method

Releases a record lock.

**Class**

`TCustomDADataSet`
Syntax

```pascal
procedure UnLock;
```

Remarks

Call the Unlock method to release the record lock made by the `Lock` method before.

Unlock is performed by rolling back to the savepoint set by the Lock method.

See Also

- `Lock`

© 1997-2021
Devart. All Rights Reserved.

5.11.1.5.4 Events

Events of the `TCustomDADataset` class.

For a complete list of the `TCustomDADataset` class members, see the `TCustomDADataset Members` topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td>BeforeUpdateExecute</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td>OnUpdateError</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
</tbody>
</table>
### OnUpdateRecord

**OnUpdateRecord** (inherited from **TMemDataSet**)

| Occurs when a single update component can not handle the updates. |

**See Also**
- **TCustomDADataSet Class**
- **TCustomDADataSet Class Members**

© 1997-2021 Devart. All Rights Reserved.

#### 5.11.1.5.4.1 AfterExecute Event

Occurs after a component has executed a query to database.

**Class**

**TCustomDADataSet**

**Syntax**

```pascal
property AfterExecute: TAfterExecuteEvent;
```

**Remarks**

Occurs after a component has executed a query to database.

**See Also**
- **TCustomDADataSet.Execute**

© 1997-2021 Devart. All Rights Reserved.

#### 5.11.1.5.4.2 AfterFetch Event

Occurs after dataset finishes fetching data from server.

**Class**

**TCustomDADataSet**

**Syntax**

```pascal
property AfterFetch: TAfterFetchEvent;
```
Remarks

The AfterFetch event occurs after dataset finishes fetching data from server.

See Also

• BeforeFetch

5.11.1.5.4.3 AfterUpdateExecute Event

Occurs after executing insert, delete, update, lock and refresh operations.

Class

TCustomDADataset

Syntax

property AfterUpdateExecute: TUpdateExecuteEvent;

Remarks

Occurs after executing insert, delete, update, lock, and refresh operations. You can use AfterUpdateExecute to set the parameters of corresponding statements.

5.11.1.5.4.4 BeforeFetch Event

Occurs before dataset is going to fetch block of records from the server.

Class

TCustomDADataset

Syntax

property BeforeFetch: TBeforeFetchEvent;

Remarks

The BeforeFetch event occurs every time before dataset is going to fetch a block of records.
from the server. Set Cancel to True to abort current fetch operation.

See Also
- AfterFetch

5.11.1.5.4.5 BeforeUpdateExecute Event

Occurs before executing insert, delete, update, lock, and refresh operations.

Class
- TCustomDADataSet

Syntax

```delphi
property BeforeUpdateExecute: TUpdateExecuteEvent;
```

Remarks

Occurs before executing insert, delete, update, lock, and refresh operations. You can use BeforeUpdateExecute to set the parameters of corresponding statements.

See Also
- AfterUpdateExecute

5.11.1.6 TCustomDASQL Class

A base class for components executing SQL statements that do not return result sets.

For a list of all members of this type, see TCustomDASQL members.

Unit
- DBAccess

Syntax

```delphi
TCustomDASQL = class(TComponent);
```
Remarks

TCustomDASQL is a base class that defines functionality for descendant classes which access database using SQL statements. Applications never use TCustomDASQL objects directly. Instead they use descendants of TCustomDASQL.

Use TCustomDASQL when client application must execute SQL statement or call stored procedure on the database server. The SQL statement should not retrieve rows from the database.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>Enables or disables changing screen cursor when executing commands in the NonBlocking mode.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return a SQL statement with expanded macros.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>Indicates the number of parameters in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to contain parameters for a SQL statement.</td>
</tr>
<tr>
<td><strong>ParamValues</strong></td>
<td>Used to get or set the values of individual field parameters that are identified by name.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>Used to indicate whether a query is prepared for execution.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.</td>
</tr>
</tbody>
</table>

---

### Methods

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BreakExec</strong></td>
<td>Breaks execution of an SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td>Checks whether TCustomDASQL still executes a SQL statement.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>Finds a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>MacroByName</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td>Finds a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
</tbody>
</table>
**UnPrepare**
Frees the resources allocated for a previously prepared query on the server and client sides.

**WaitExecuting**
Waits until TCustomDASQL executes a SQL statement.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a SQL statement has been executed.</td>
</tr>
</tbody>
</table>

5.11.1.6.2 Properties

Properties of the TCustomDASQL class.

For a complete list of the TCustomDASQL class members, see the TCustomDASQL Members topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>Enables or disables changing screen cursor when executing commands in the NonBlocking mode.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return a SQL statement with expanded macros.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Macros</td>
<td>Macros property. Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>Indicates the number of parameters in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to contain parameters for a SQL statement.</td>
</tr>
<tr>
<td>ParamValues</td>
<td>Used to get or set the values of individual field parameters that are identified by name.</td>
</tr>
<tr>
<td>Prepared</td>
<td>Used to indicate whether a query is prepared for execution.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.</td>
</tr>
</tbody>
</table>

See Also
- TCustomDASQL Class
- TCustomDASQL Class Members

Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
Class

TCustomDASQL

Syntax

property ChangeCursor: boolean;

Remarks

Set the ChangeCursor property to False to prevent the screen cursor from changing to crSQLArrow when executing commands in the NonBlocking mode. The default value is True.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.6.2.2 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

TCustomDASQL

Syntax

property Connection: TCustomDAConnection;

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

© 1997-2021 Devart. All Rights Reserved.
5.11.1.6.2.3 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

TCustomDASQL

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the PgDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

**Note:** If TPgSQLMonitor is used in the project and the TPgSQLMonitor.Active property is set to False, the debug window is not displayed.

See Also

- TCustomDADataset.Debug

5.11.1.6.2.4 FinalSQL Property

Used to return a SQL statement with expanded macros.

Class

TCustomDASQL

Syntax

```
property FinalSQL: string;
```

Remarks

Read the FinalSQL property to return a SQL statement with expanded macros. This is the exact statement that will be passed on to the database server.
5.11.1.6.2.5  MacroCount Property

Used to get the number of macros associated with the Macros property.

Class

TCustomDASQL

Syntax

```delphi
property MacroCount: word;
```

Remarks

Use the MacroCount property to get the number of macros associated with the Macros property.

See Also

- Macros

5.11.1.6.2.6  Macros Property

Makes it possible to change SQL queries easily.

Class

TCustomDASQL

Syntax

```delphi
property Macros: TMacros stored False;
```

Remarks

With the help of macros you can easily change SQL query text at design- or runtime. Marcos extend abilities of parameters and allow to change conditions in a WHERE clause or sort order in an ORDER BY clause. You just insert &MacroName in the SQL query text and change value of macro in the Macro property editor at design time or call the MacroByName
function at run time. At the time of opening the query macro is replaced by its value.

See Also
- **TMacro**
- **MacroByName**
- **Params**

### 5.11.1.6.2.7 ParamCheck Property

Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.

**Class**

**TCustomDASQL**

**Syntax**

```property ParamCheck: boolean default True;```

**Remarks**

Use the ParamCheck property to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.

Set ParamCheck to True to let TCustomDASQL generate the Params property for the dataset based on a SQL statement automatically.

Setting ParamCheck to False can be used if the dataset component passes to a server the DDL statements that contain, for example, declarations of the stored procedures that will accept parameterized values themselves. The default value is True.

See Also
- **Params**
5.11.6.2.8  ParamCount Property

Indicates the number of parameters in the Params property.

Class

TCustomDASQL

Syntax

```
property ParamCount: word;
```

Remarks

Use the ParamCount property to determine how many parameters are there in the Params property.

5.11.6.2.9  Params Property

Used to contain parameters for a SQL statement.

Class

TCustomDASQL

Syntax

```
property Params: TDAParams stored False;
```

Remarks

Access the Params property at runtime to view and set parameter names, values, and data types dynamically (at design-time use the Parameters editor to set parameter properties). Params is a zero-based array of parameter records. Index specifies the array element to access. An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

Example

Setting parameters at runtime:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with PgSQL do
```
begin
    SQL.Clear;
    SQL.Add('INSERT INTO Temp_Table(Id, Name)');
    SQL.Add('VALUES (:id, :Name)');
    ParamByName('Id').AsInteger := 55;
    Params[1].AsString := 'Green';
    Execute;
end;

See Also

- TDAParam
- FindParam
- Macros

© 1997-2021
Devart. All Rights Reserved.

5.11.6.2.10 ParamValues Property(Indexer)

Used to get or set the values of individual field parameters that are identified by name.

Class

TCustomDASQL

Syntax

```property ParamValues[const ParamName: string]: Variant; default;```

Parameters

- **ParamName**: Holds parameter names separated by semicolon.

Remarks

Use the ParamValues property to get or set the values of individual field parameters that are identified by name.

Setting ParamValues sets the Value property for each parameter listed in the ParamName string. Specify the values as Variants.

Getting ParamValues retrieves an array of variants, each of which represents the value of one of the named parameters.

**Note**: The Params array is generated implicitly if ParamCheck property is set to True. If ParamName includes a name that does not match any of the parameters in Items, an
exception is raised.

5.11.1.6.2.11 Prepared Property

Used to indicate whether a query is prepared for execution.

Class

TCustomDASQL

Syntax

```property`` Prepared: boolean;

Remarks

Check the Prepared property to determine if a query is already prepared for execution. True means that the query has already been prepared. As a rule prepared queries are executed faster, but the preparation itself also takes some time. One of the proper cases for using preparation is parametrized queries that are executed several times.

See Also

• Prepare

5.11.1.6.2.12 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

TCustomDASQL

Syntax

```property`` RowsAffected: integer;

Remarks
Check `RowsAffected` to determine how many rows were inserted, updated, or deleted during the last query operation. If `RowsAffected` is -1, the query has not inserted, updated, or deleted any rows.

5.11.1.6.2.13 SQL Property

Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

Class

TCustomDASQL

Syntax

```property
SQL: TStrings;
```

Remarks

Use the SQL property to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called. At design time the SQL property can be edited by invoking the String List editor in Object Inspector.

See Also

- `FinalSQL`
- `TCustomDASQL.Execute`

5.11.1.6.3 Methods

Methods of the TCustomDASQL class.

For a complete list of the TCustomDASQL class members, see the TCustomDASQL Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BreakExec</td>
<td>Breaks execution of an SQL statement on the server.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>Checks whether TCustomDASQL still executes a SQL statement.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindParam</td>
<td>Finds a parameter with the specified name.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Finds a parameter with the specified name.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>WaitExecuting</td>
<td>Waits until TCustomDASQL executes a SQL statement.</td>
</tr>
</tbody>
</table>

See Also
- TCustomDASQL Class
- TCustomDASQL Class Members

© 1997-2021 Devart. All Rights Reserved.  
Request Support  DAC Forum  Provide Feedback

5.11.1.6.3.1 BreakExec Method

Breaks execution of an SQL statement on the server.

Class

TCustomDASQL

Syntax

```procedure BreakExec;```
Remarks

Call the BreakExec method to break execution of an SQL statement on the server. It makes sense to call BreakExec only from another thread. Useful when NonBlocking is True.

See Also
- TCustomDASQL.Execute
- TCustomDADataSet.BreakExec

Executes a SQL statement on the server.

Class

TCustomDASQL

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Execute(Iter: integer; Offset: integer)</td>
<td>Used to perform Batch operations.</td>
</tr>
</tbody>
</table>

Executes a SQL statement on the server.

Class

TCustomDASQL

Syntax

procedure Execute; overload; virtual;

Remarks

Call the Execute method to execute a SQL statement on the server. If the SQL statement has OUT parameters, use the TCustomDASQL.ParamByName method or the
**TCustomDASQL.Params** property to get their values. **Iters** argument specifies the number of times this statement is executed for the DML array operations.

Used to perform **Batch operations**.

**Class**

**TCustomDASQL**

**Syntax**

```procedure Execute(Iters: integer; Offset: integer = 0); overload;
virtual;```

**Parameters**

- **Iters**
  - Specifies the number of inserted rows.

- **Offset**
  - Points the array element, which the Batch operation starts from. 0 by default.

**Remarks**

The Execute method executes the specified batch SQL query. See the **Batch operations** article for samples.

**See Also**

- **Batch operations**

5.11.1.6.3.3 Executing Method

Checks whether **TCustomDASQL** still executes a SQL statement.

**Class**

**TCustomDASQL**

**Syntax**
function Executing: boolean;

Return Value
True, if a SQL statement is still being executed by TCustomDASQL.

Remarks
Check Executing to find out whether TCustomDASQL still executes a SQL statement.

5.11.1.6.3.4 FindMacro Method

Finds a macro with the specified name.

Class
TCustomDASQL

Syntax
function FindMacro(const Value: string): TMacro;

Parameters
Value
Holds the name of a macro to search for.

Return Value
TMacro object if a match is found, nil otherwise.

Remarks
Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the `TMacros.Items` property to avoid depending on the order of the items.

See Also
- TMacro
- Macros
- MacroByName

© 1997-2021
Devart. All Rights Reserved.
FindParam Method

Finds a parameter with the specified name.

Class

TCustomDASQL

Syntax

```pascal
function FindParam(const Value: string): TDAParam;
```

Parameters

Value

Holds the parameter name to search for.

Return Value

a TDAParam object, if a parameter with the specified name has been found. If it has not, returns nil.

Remarks

Call the FindParam method to find a parameter with the specified name in a dataset.

See Also

- ParamByName

MacroByName Method

Finds a macro with the specified name.

Class

TCustomDASQL

Syntax

```pascal
function MacroByName(const Value: string): TMacro;
```

Parameters

Value

Holds the name of a macro to search for.

Return Value
TMacro object if a match is found.

Remarks
Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

See Also
• TMacro
• Macros
• FindMacro

© 1997-2021 Devart. All Rights Reserved.

5.11.1.6.3.7 ParamByName Method

Finds a parameter with the specified name.

Class
TCustomDASQL

Syntax

```
function ParamByName(const Value: string): TDAParam;
```

Parameters

Value
Holds the name of the parameter to search for.

Return Value
a TDAParam object, if a match was found. Otherwise, an exception is raised.

Remarks
Use the ParamByName method to find a parameter with the specified name. If no parameter with the specified name found, an exception is raised.
5.11.1.6.3.8  Prepare Method

Allocates, opens, and parses cursor for a query.

Class

TCustomDASQL

Syntax

```
procedure Prepare; virtual;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- Prepared
- UnPrepare

© 1997-2021
Devart. All Rights Reserved.
5.11.1.6.3.9 UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class

TCustomDASQL

Syntax

procedure UnPrepare; virtual;

Remarks

Call the UnPrepare method to free resources allocated for a previously prepared query on the server and client sides.

See Also

• Prepare

© 1997-2021 Devart. All Rights Reserved.

5.11.1.6.3.10 WaitExecuting Method

Waits until TCustomDASQL executes a SQL statement.

Class

TCustomDASQL

Syntax

function WaitExecuting(TimeOut: integer = 0): boolean;

Parameters

TimeOut

Holds the time in seconds to wait while TCustomDASQL executes the SQL statement. Zero means infinite time.

Return Value

True, if the execution of a SQL statement was completed in the preset time.

Remarks

Call the WaitExecuting method to wait until TCustomDASQL executes a SQL statement.
5.11.1.6.4 Events

Events of the TCustomDASQL class.

For a complete list of the TCustomDASQL class members, see the TCustomDASQL Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a SQL statement has been executed.</td>
</tr>
</tbody>
</table>

See Also

- Executing
- TCustomDASQL Class
- TCustomDASQL Class Members

5.11.1.6.4.1 AfterExecute Event

Occurs after a SQL statement has been executed.

Class

TCustomDASQL

Syntax

```pascal
property AfterExecute: TAfterExecuteEvent;
```

Remarks

Occurs after a SQL statement has been executed. This event may be used for descendant components which use multithreaded environment.
5.11.1.7  TCustomDAUpdateSQL Class

A base class for components that provide DML statements for more flexible control over data modifications.

For a list of all members of this type, see TCustomDAUpdateSQL members.

Unit

DBAccess

Syntax

TCustomDAUpdateSQL = class(TComponent);

Remarks

TCustomDAUpdateSQL is a base class for components that provide DML statements for more flexible control over data modifications. Besides providing BDE compatibility, this component allows to associate a separate component for each update command.

See Also

- TCustomPgDataSet.UpdateObject

Members

TCustomDAUpdateSQL class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSet</td>
<td>Used to hold a reference to the TCustomDADataset object that is being updated.</td>
</tr>
</tbody>
</table>
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apply</strong></td>
<td>Sets parameters for a SQL statement and executes it to update a record.</td>
</tr>
<tr>
<td><strong>ExecSQL</strong></td>
<td>Executes a SQL statement.</td>
</tr>
</tbody>
</table>
Properties of the `TCustomDAUpdateSQL` class.

For a complete list of the `TCustomDAUpdateSQL` class members, see the `TCustomDAUpdateSQL Members` topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSet</td>
<td>Used to hold a reference to the <code>TCustomDADataset</code> object that is being updated.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to return a SQL statement for one of the <code>ModifySQL</code>, <code>InsertSQL</code>, or <code>DeleteSQL</code> properties.</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteObject</td>
<td>Provides ability to perform advanced adjustment of the delete operations.</td>
</tr>
<tr>
<td>DeleteSQL</td>
<td>Used when deleting a record.</td>
</tr>
<tr>
<td>InsertObject</td>
<td>Provides ability to perform advanced adjustment of insert operations.</td>
</tr>
<tr>
<td>InsertSQL</td>
<td>Used when inserting a record.</td>
</tr>
<tr>
<td>LockObject</td>
<td>Provides ability to perform advanced adjustment of lock operations.</td>
</tr>
<tr>
<td>LockSQL</td>
<td>Used to lock the current record.</td>
</tr>
<tr>
<td>ModifyObject</td>
<td>Provides ability to perform advanced adjustment of modify operations.</td>
</tr>
<tr>
<td>ModifySQL</td>
<td>Used when updating a record.</td>
</tr>
<tr>
<td>RefreshObject</td>
<td>Provides ability to perform</td>
</tr>
</tbody>
</table>
advanced adjustment of refresh operations.

**RefreshSQL**

Used to specify an SQL statement that will be used for refreshing the current record by `TCustomDADataset.RefreshRecord` procedure.

---

**See Also**
- `TCustomDAUpdateSQL Class`
- `TCustomDAUpdateSQL Class Members`

---

5.11.1.7.2.1  **DataSet Property**

Used to hold a reference to the TCustomDADataset object that is being updated.

**Class**

`TCustomDAUpdateSQL`

**Syntax**

```
property DataSet: TCustomDADataset;
```

**Remarks**

The DataSet property holds a reference to the TCustomDADataset object that is being updated. Generally it is not used directly.

---

5.11.1.7.2.2  **DeleteObject Property**

Provides ability to perform advanced adjustment of the delete operations.

**Class**

`TCustomDAUpdateSQL`
Syntax

property DeleteObject: TComponent;

Remarks

Assign SQL component or a TCustomPgDataSet descendant to this property to perform advanced adjustment of the delete operations. In some cases this can give some additional performance. Use the same principle to set the SQL property of an object as for setting the DeleteSQL property.

See Also

- DeleteSQL

© 1997-2021 Devart. All Rights Reserved.

5.11.1.7.2.3 DeleteSQL Property

Used when deleting a record.

Class

TCustomDAUpdateSQL

Syntax

property DeleteSQL: TStrings;

Remarks

Set the DeleteSQL property to a DELETE statement to use when deleting a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.7.2.4 InsertObject Property

Provides ability to perform advanced adjustment of insert operations.

Class
**TCustomDAUpdateSQL**

**Syntax**

```pascal
property InsertObject: TComponent;
```

**Remarks**

Assign SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of insert operations. In some cases this can give some additional performance. Set the SQL property of the object in the same way as used for the `InsertSQL` property.

**See Also**
- `InsertSQL`

© 1997-2021 Devart. All Rights Reserved.

---

**5.11.1.7.2.5 InsertSQL Property**

Used when inserting a record.

**Class**

**TCustomDAUpdateSQL**

**Syntax**

```pascal
property InsertSQL: TStrings;
```

**Remarks**

Set the `InsertSQL` property to an INSERT INTO statement to use when inserting a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.
5.11.1.7.2.6 LockObject Property

Provides ability to perform advanced adjustment of lock operations.

Class

**TCustomDAUpdateSQL**

Syntax

```
property LockObject: TComponent;
```

Remarks

Assign a SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of lock operations. In some cases that can give some additional performance. Set the SQL property of an object in the same way as used for the LockSQL property.

See Also

- LockSQL

© 1997-2021 Devart. All Rights Reserved.

5.11.1.7.2.7 LockSQL Property

Used to lock the current record.

Class

**TCustomDAUpdateSQL**

Syntax

```
property LockSQL: TStrings;
```

Remarks

Use the LockSQL property to lock the current record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.
5.11.1.7.2.8 ModifyObject Property

Provides ability to perform advanced adjustment of modify operations.

Class

TCustomDAUpdateSQL

Syntax

\texttt{property \ ModifyObject: TComponent;}

Remarks

Assign a SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of modify operations. In some cases this can give some additional performance. Set the SQL property of the object in the same way as used for the ModifySQL property.

See Also

• ModifySQL

5.11.1.7.2.9 ModifySQL Property

Used when updating a record.

Class

TCustomDAUpdateSQL

Syntax

\texttt{property \ ModifySQL: TStrings;}

Remarks

Set ModifySQL to an UPDATE statement to use when updating a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.
5.11.1.7.2.10 RefreshObject Property

Provides ability to perform advanced adjustment of refresh operations.

Class

TCustomDAUpdateSQL

Syntax

property RefreshObject: TComponent;

Remarks

Assign a SQL component or TCustomPgDataSet descendant to this property to perform advanced adjustment of refresh operations. In some cases that can give some additional performance. Set the SQL property of the object in the same way as used for the RefreshSQL property.

See Also

• RefreshSQL

5.11.1.7.2.11 RefreshSQL Property

Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataSet.RefreshRecord procedure.

Class

TCustomDAUpdateSQL

Syntax

property RefreshSQL: TStrings;

Remarks

Use the RefreshSQL property to specify a SQL statement that will be used for refreshing the current record by the TCustomDADataSet.RefreshRecord procedure.

You can assign to SQLRefresh a WHERE clause only. In such a case it is added to SELECT defined by the SQL property by TCustomDADataSet.AddWhere.
To create a RefreshSQL statement at design time, use the query statements editor.

See Also
- TCustomDADataSet.RefreshRecord

Class
TCustomDAUpdateSQL

Syntax

```property SQL[UpdateKind: TUpdateKind]: TStrings;```

Parameters

`UpdateKind`
Specifies which of update SQL statements to return.

Remarks

Returns a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties, depending on the value of the UpdateKind index.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply</td>
<td>Sets parameters for a SQL</td>
</tr>
</tbody>
</table>
Sets parameters for a SQL statement and executes it to update a record.

Class

**TCustomDAUpdateSQL**

Syntax

```plaintext
procedure Apply(UpdateKind: TUpdateKind); virtual;
```

Parameters

*UpdateKind*

Specifies which of update SQL statements to execute.

Remarks

Call the Apply method to set parameters for a SQL statement and execute it to update a record. UpdateKind indicates which SQL statement to bind and execute.

Apply is primarily intended for manually executing update statements from an OnUpdateRecord event handler.

**Note:** If a SQL statement does not contain parameters, it is more efficient to call **ExecSQL** instead of Apply.

See Also

- **ExecSQL**

© 1997-2021

Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback
5.11.1.7.3.2 ExecSQL Method

Executes a SQL statement.

Class

`TCustomDAUpdateSQL`

Syntax

```plaintext
procedure ExecSQL(UpdateKind: TUpdateKind);
```

Parameters

`UpdateKind`

Specifies the kind of update statement to be executed.

Remarks

Call the ExecSQL method to execute a SQL statement, necessary for updating the records belonging to a read-only result set when cached updates is enabled. `UpdateKind` specifies the statement to execute.

ExecSQL is primarily intended for manually executing update statements from the `OnUpdateRecord` event handler.

**Note:** To both bind parameters and execute a statement, call `Apply`.

See Also

- `Apply`

© 1997-2021 Devart. All Rights Reserved.

5.11.1.8 TDACCondition Class

 Represents a condition from the `TDAConditions` list.

For a list of all members of this type, see `TDACCondition` members.

Unit

`DBAccess`

Syntax

```plaintext
TDACCondition = class(TCollectionItem);
```
Remarks

Manipulate conditions using `TDAConditions`.

See Also

- `TDAConditions`

© 1997-2021
Devart. All Rights Reserved.

### 5.11.1.8.1 Members

**TDACondition** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Indicates whether the condition is enabled or not</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the condition</td>
</tr>
<tr>
<td>Value</td>
<td>The value of the condition</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>Disables the condition</td>
</tr>
<tr>
<td>Enable</td>
<td>Enables the condition</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.

### 5.11.1.8.2 Properties

Properties of the `TDACondition` class.

For a complete list of the `TDACondition` class members, see the `TDACondition Members` topic.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Indicates whether the condition is enabled or not</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the condition</td>
</tr>
<tr>
<td>Value</td>
<td>The value of the condition</td>
</tr>
</tbody>
</table>

See Also
- **TDACCondition Class**
- **TDACCondition Class Members**

5.11.1.8.2.1 Enabled Property

Indicates whether the condition is enabled or not

**Class**

**TDACCondition**

**Syntax**

```plaintext
property Enabled: Boolean default True;
```

5.11.1.8.2.2 Name Property

The name of the condition

**Class**

**TDACCondition**

**Syntax**

```plaintext
property Name: string;
```
5.11.1.8.3 Value Property

The value of the condition

Class

**TDACCondition**

Syntax

```plaintext
property value: string;
```

5.11.1.8.3 Methods

Methods of the **TDACCondition** class.

For a complete list of the **TDACCondition** class members, see the [TDACCondition Members](#) topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>Disables the condition</td>
</tr>
<tr>
<td>Enable</td>
<td>Enables the condition</td>
</tr>
</tbody>
</table>

See Also

- [TDACCondition Class](#)
- [TDACCondition Class Members](#)

© 1997-2021 Devart. All Rights Reserved.
5.11.1.8.3.2 Enable Method

Enables the condition

Class

TDACondition

Syntax

**procedure** Enable;

5.11.1.9 TDAConditions Class

Holds a collection of TDACondition objects.

For a list of all members of this type, see TDAConditions members.

Unit

DBAccess

Syntax

TDAConditions = **class**(TCollection);

Remarks

The given example code

```
UniTable1.Conditions.Add('1','JOB="MANAGER"');
UniTable1.Conditions.Add('2','SAL>2500');
UniTable1.Conditions.Enable;
UniTable1.Open;
```

will return the following SQL:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and
```
5.11.1.9.1 Members

**TDAConditions** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>Used to iterate through all the conditions.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Indicates whether the condition is enabled.</td>
</tr>
<tr>
<td>Items</td>
<td>Used to iterate through all conditions.</td>
</tr>
<tr>
<td>Text</td>
<td>The property returns condition names and values as CONDITION_NAME=CONDITION</td>
</tr>
<tr>
<td>WhereSQL</td>
<td>Returns the SQL WHERE condition added in the Conditions property.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Overloaded. Adds a condition to the WHERE clause of the query.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the condition.</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables the condition.</td>
</tr>
<tr>
<td>Enable</td>
<td>Enables the condition.</td>
</tr>
<tr>
<td>Find</td>
<td>Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.</td>
</tr>
<tr>
<td>Get</td>
<td>Retrieving a TDACondition object by its name. If found,</td>
</tr>
</tbody>
</table>
5.11.1.9.2 Properties

Properties of the **TDAConditions** class.

For a complete list of the **TDAConditions** class members, see the [TDAConditions Members](#) topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>Used to iterate through all the conditions.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Indicates whether the condition is enabled</td>
</tr>
<tr>
<td>Items</td>
<td>Used to iterate through all conditions.</td>
</tr>
<tr>
<td>Text</td>
<td>The property returns condition names and values as <code>CONDITION_NAME=COND ITION</code></td>
</tr>
<tr>
<td>WhereSQL</td>
<td>Returns the SQL WHERE condition added in the Conditions property.</td>
</tr>
</tbody>
</table>

See Also
- [TDAConditions Class](#)
- [TDAConditions Class Members](#)
5.11.1.9.2.1 Condition Property (Indexer)

Used to iterate through all the conditions.

Class

TDAConditions

Syntax

| property | Condition[Index: Integer]: TDACondition; |

Parameters

Index

© 1997-2021

5.11.1.9.2.2 Enabled Property

Indicates whether the condition is enabled

Class

TDAConditions

Syntax

| property | Enabled: Boolean; |

© 1997-2021

5.11.1.9.2.3 Items Property (Indexer)

Used to iterate through all conditions.

Class

TDAConditions

Syntax

| property | Items[Index: Integer]: TDACondition; default; |
Parameters

Index
Holds an index in the range 0..Count - 1.

Remarks
Use the Items property to iterate through all conditions. Index identifies the index in the range 0..Count - 1. Items can reference a particular condition by its index, but the Condition property is preferred in order to avoid depending on the order of the conditions.

5.11.1.9.2.4 Text Property
The property returns condition names and values as CONDITION_NAME=CONDITION

Class
TDAConditions

Syntax

property Text: string;

5.11.1.9.2.5 WhereSQL Property
Returns the SQL WHERE condition added in the Conditions property.

Class
TDAConditions

Syntax

property WhereSQL: string;
Methods of the **TDAConditions** class.

For a complete list of the **TDAConditions** class members, see the [TDAConditions Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add</strong></td>
<td>Overloaded. Adds a condition to the WHERE clause of the query.</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>Deletes the condition</td>
</tr>
<tr>
<td><strong>Disable</strong></td>
<td>Disables the condition</td>
</tr>
<tr>
<td><strong>Enable</strong></td>
<td>Enables the condition</td>
</tr>
<tr>
<td><strong>Find</strong></td>
<td>Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.</td>
</tr>
<tr>
<td><strong>Get</strong></td>
<td>Retrieving a TDACondition object by its name. If found, the TDACondition object is returned, otherwise - an exception is raised.</td>
</tr>
<tr>
<td><strong>IndexOf</strong></td>
<td>Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.</td>
</tr>
<tr>
<td><strong>Remove</strong></td>
<td>Removes the condition</td>
</tr>
</tbody>
</table>

**See Also**
- [TDAConditions Class](#)
- [TDAConditions Class Members](#)
5.11.1.9.3.1 Add Method

Adds a condition to the WHERE clause of the query.

Class

TDAConditions

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add(const Value: string; Enabled: Boolean)</td>
<td>Adds a condition to the WHERE clause of the query.</td>
</tr>
<tr>
<td>Add(const Name: string; const Value: string; Enabled: Boolean)</td>
<td>Adds a condition to the WHERE clause of the query.</td>
</tr>
</tbody>
</table>

Adds a condition to the WHERE clause of the query.

Class

TDAConditions

Syntax

```c
function Add(const Value: string; Enabled: Boolean = True): TDACondition; overload;
```

Parameters

- **Value**
  - The value of the condition

- **Enabled**
  - Indicates that the condition is enabled

Remarks

If you want then to access the condition, you should use `Add` and its name in the Name parameter.

The given example code will return the following SQL:

```sql
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and
```
Adds a condition to the WHERE clause of the query.

Class

**TDAConditions**

Syntax

```go
function Add(const Name: string; const Value: string; Enabled: Boolean = True): TDACondition; overload;
```

Parameters

- **Name**
  - Sets the name of the condition
- **Value**
  - The value of the condition
- **Enabled**
  - Indicates that the condition is enabled

Remarks

The given example code will return the following SQL:

```sql
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and (SAL<2500)
```

5.11.1.9.3.2  Delete Method

Deletes the condition

Class

**TDAConditions**

Syntax

```go
procedure Delete(Index: integer);
```
Parameters

Index

5.11.1.9.3.3 Disable Method

Disables the condition

Class

TDAConditions

Syntax

procedure Disable;

5.11.1.9.3.4 Enable Method

Enables the condition

Class

TDAConditions

Syntax

procedure Enable;

5.11.1.9.3.5 Find Method

Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.

Class

TDAConditions
5.11.1.9.3.6 Get Method

Retrieving a `TDACCondition` object by its name. If found, the `TDACCondition` object is returned, otherwise - an exception is raised.

Class

`TDACConditions`

Syntax

```pascal
function Find(const Name: string): TDACCondition;
```

Parameters

Name

© 1997-2021 Devart. All Rights Reserved.

5.11.1.9.3.7 IndexOf Method

Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.

Class

`TDACConditions`

Syntax

```pascal
function IndexOf(const Name: string): Integer;
```

Parameters

Name

© 1997-2021 Devart. All Rights Reserved.
5.11.1.9.3.8 Remove Method

Removes the condition

Class
TDAConditions

Syntax

```pascal
procedure Remove(const Name: string);
```

Parameters

Name
Specifies the name of the removed condition

5.11.1.10 TDACreationOptions Class

This class allows setting up the behaviour of the TDAConnection class.

For a list of all members of this type, see TDACreationOptions members.

Unit

DBAccess

Syntax

```pascal
TDACreationOptions = class(TPersistent);
```

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 1997-2021 Devart. All Rights Reserved.
<table>
<thead>
<tr>
<th>Public</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>DefaultSortType</td>
<td>Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.FieldNames property of a dataset.</td>
</tr>
<tr>
<td>DisconnectedMode</td>
<td>Used to open a connection only when needed for performing a server call and closes after performing the operation.</td>
</tr>
<tr>
<td>KeepDesignConnected</td>
<td>Used to prevent an application from establishing a connection at the time of startup.</td>
</tr>
<tr>
<td>LocalFailover</td>
<td>If True, the TCustomDACConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
</tbody>
</table>

Properties of the **TDACConnectionOptions** class.

For a complete list of the **TDACConnectionOptions** class members, see the **TDACConnectionOptions Members** topic.
<table>
<thead>
<tr>
<th>Published</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>AllowImplicitConnect</td>
<td>Specifies whether to allow or not implicit connection opening.</td>
</tr>
</tbody>
</table>

**DisconnectedMode**
Used to open a connection only when needed for performing a server call and closes after performing the operation.

**KeepDesignConnected**
Used to prevent an application from establishing a connection at the time of startup.

**LocalFailover**
If True, the TCustomDACConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

See Also
- TDAConnectionOptions Class
- TDAConnectionOptions Class Members

© 1997-2021 Devart. All Rights Reserved.

5.11.1.10.2.1 AllowImplicitConnect Property

Specifies whether to allow or not implicit connection opening.

Class
TDAConnectionOptions

Syntax
**property** AllowImplicitConnect: boolean default True;

**Remarks**

Use the AllowImplicitConnect property to specify whether allow or not implicit connection opening.

If a closed connection has AllowImplicitConnect set to True and a dataset that uses the connection is opened, the connection is opened implicitly to allow opening the dataset.

If a closed connection has AllowImplicitConnect set to False and a dataset that uses the connection is opened, an exception is raised.

The default value is True.

5.11.1.10.2.2 DefaultSortType Property

Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the **TMemDataSet.IndexFieldNames** property of a dataset.

**Class**

**TDACConnectionOptions**

**Syntax**

**property** DefaultSortType: TSortType default stCaseSensitive;

**Remarks**

Use the DefaultSortType property to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the **TMemDataSet.IndexFieldNames** property of a dataset.

© 1997-2021 Devart. All Rights Reserved.
Class
TDACConnectionOptions

Syntax

```pascal
property DisconnectedMode: boolean default False;
```

Remarks
If True, connection opens only when needed for performing a server call and closes after performing the operation. Datasets remain opened when connection closes. May be useful to save server resources and operate in unstable or expensive network. Drawback of using disconnect mode is that each connection establishing requires some time for authorization. If connection is often closed and opened it can slow down the application work. See the Disconnected Mode topic for more information.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.10.2.4 KeepDesignConnected Property

Used to prevent an application from establishing a connection at the time of startup.

Class
TDACConnectionOptions

Syntax

```pascal
property KeepDesignConnected: boolean default True;
```

Remarks
At the time of startup prevents application from establishing a connection even if the Connected property was set to True at design-time. Set KeepDesignConnected to False to initialize the connected property to False, even if it was True at design-time.

© 1997-2021 Devart. All Rights Reserved.
5.11.1.10.2.5 LocalFailover Property

If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

Class

TDAClass

Syntax

property LocalFailover: boolean default False;

Remarks

If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks. Read the Working in an Unstable Network topic for more information about using failover.

5.11.11 TDAClass

This class is used to set up the SSL options.

For a list of all members of this type, see TDAClass members.

Unit

DBAccess

Syntax

TDAClass = class (TPersistent);

© 1997-2021 Devart. All Rights Reserved.
### 5.11.1.11.2 Properties

Properties of the `TDACConnectionSSLOptions` class.

For a complete list of the `TDACConnectionSSLOptions` class members, see the `TDACConnectionSSLOptions Members` topic.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACert</td>
<td>Holds the path to the certificate authority file.</td>
</tr>
<tr>
<td>Cert</td>
<td>Holds the path to the client certificate.</td>
</tr>
<tr>
<td>CipherList</td>
<td>Holds the list of allowed SSL ciphers.</td>
</tr>
<tr>
<td>Key</td>
<td>Holds the path to the private client key.</td>
</tr>
</tbody>
</table>

### See Also
- `TDACConnectionSSLOptions Class`
- `TDACConnectionSSLOptions Class Members`
5.11.1.11.1.1  CACert Property

Holds the path to the certificate authority file.

**Class**

**TDAConnectionSSLOptions**

**Syntax**

```
property CACert: string;
```

**Remarks**

Use the CACert property to specify the path to the certificate authority file.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.11.2.2  Cert Property

Holds the path to the client certificate.

**Class**

**TDAConnectionSSLOptions**

**Syntax**

```
property Cert: string;
```

**Remarks**

Use the Cert property to specify the path to the client certificate.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.11.2.3  CipherList Property

Holds the list of allowed SSL ciphers.

**Class**

**TDAConnectionSSLOptions**
Syntax

```delphi
property CipherList: string;
```

Remarks
Use the CipherList property to specify the list of allowed SSL ciphers.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.11.2.4 Key Property

Holds the path to the private client key.

Class

`TDACConnectionSSLOptions`

Syntax

```delphi
property Key: string;
```

Remarks
Use the Key property to specify the path to the private client key.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.12 `TDADatasetOptions` Class

This class allows setting up the behaviour of the TDADataset class.

For a list of all members of this type, see `TDADatasetOptions` members.

Unit

`DBAccess`

Syntax

```delphi
TDADatasetOptions = class(TPersistent);
```

© 1997-2021
Devart. All Rights Reserved.
### TDADataSetOptions class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPrepare</td>
<td>Used to execute automatic TCustomDADataSet.Prepare on the query execution.</td>
</tr>
<tr>
<td>CacheCalcFields</td>
<td>Used to enable caching of the TField.Calculated and TField.Lookup fields.</td>
</tr>
<tr>
<td>CompressBlobMode</td>
<td>Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td>DefaultValues</td>
<td>Used to request default values/expressions from the server and assign them to the DefaultExpression property.</td>
</tr>
<tr>
<td>DetailDelay</td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td>FieldsOrigin</td>
<td>Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.</td>
</tr>
<tr>
<td>FlatBuffers</td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td>InsertAllSetFields</td>
<td>Used to include all set dataset fields in the generated INSERT statement.</td>
</tr>
<tr>
<td>LocalMasterDetail</td>
<td>Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LongStrings</td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td>MasterFieldsNullable</td>
<td>Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).</td>
</tr>
<tr>
<td>NumberRange</td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.</td>
</tr>
<tr>
<td>QueryRecCount</td>
<td>Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td>RemoveOnRefresh</td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td>RequiredFields</td>
<td>Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td>ReturnParams</td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td>SetFieldsReadOnly</td>
<td>Used for a dataset to set the ReadOnly property to True for all fields that do not refer to the server.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AutoPrepare</td>
<td>Used to execute automatic TCustomDADataSet_Prepare on the query execution.</td>
</tr>
<tr>
<td>CacheCalcFields</td>
<td>Used to enable caching of the TField.Calculated and TField.Lookup fields.</td>
</tr>
<tr>
<td>CompressBlobMode</td>
<td>Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td><strong>DefaultValues</strong></td>
<td>Used to request default values/expressions from the server and assign them to the DefaultExpression property.</td>
</tr>
<tr>
<td><strong>DetailDelay</strong></td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td><strong>FieldsOrigin</strong></td>
<td>Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.</td>
</tr>
<tr>
<td><strong>FlatBuffers</strong></td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td><strong>InsertAllSetFields</strong></td>
<td>Used to include all set dataset fields in the generated INSERT statement.</td>
</tr>
<tr>
<td><strong>LocalMasterDetail</strong></td>
<td>Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.</td>
</tr>
<tr>
<td><strong>LongStrings</strong></td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td><strong>MasterFieldsNullable</strong></td>
<td>Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).</td>
</tr>
<tr>
<td><strong>NumberRange</strong></td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate...</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>QueryRecCount</strong></td>
<td>Used for TCustomDADataset to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td><strong>QuoteNames</strong></td>
<td>Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td><strong>RemoveOnRefresh</strong></td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td><strong>RequiredFields</strong></td>
<td>Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td><strong>ReturnParams</strong></td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td><strong>SetFieldsReadOnly</strong></td>
<td>Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdATING TABLE or can not be updated.</td>
</tr>
<tr>
<td><strong>StrictUpdate</strong></td>
<td>Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.</td>
</tr>
<tr>
<td><strong>TrimFixedChar</strong></td>
<td>Specifies whether to discard all trailing spaces in the string fields of a dataset.</td>
</tr>
<tr>
<td><strong>UpdateAllFields</strong></td>
<td>Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
</tbody>
</table>
**UpdateBatchSize**

Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

See Also
- [TDADatasetOptions Class](#)
- [TDADatasetOptions Class Members](#)

---

**AutoPrepare Property**

Used to execute automatic [TCustomDADataSet.Prepare](#) on the query execution.

**Class**

[TDADataSetOptions](#)

**Syntax**

```property
AutoPrepare: boolean default False;
```

**Remarks**

Use the AutoPrepare property to execute automatic [TCustomDADataSet.Prepare](#) on the query execution. Makes sense for cases when a query will be executed several times, for example, in Master/Detail relationships.

---

**CacheCalcFields Property**

Used to enable caching of the TField.Calculated and TField.Lookup fields.

**Class**

[TDADataSetOptions](#)

**Syntax**

---

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback
**property** CacheCalcFields: boolean **default** False;

**Remarks**

Use the CacheCalcFields property to enable caching of the TField.Calculated and TField.Lookup fields. It can be useful for reducing CPU usage for calculated fields. Using caching of calculated and lookup fields increases memory usage on the client side.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.12.2.3 CompressBlobMode Property

Used to store values of the BLOB fields in compressed form.

**Class**

**TDADataSetOptions**

**Syntax**

**property** CompressBlobMode: TCompressBlobMode **default** cbNone;

**Remarks**

Use the CompressBlobMode property to store values of the BLOB fields in compressed form. Add the MemData unit to uses list to use this option. Compression rate greatly depends on stored data, for example, usually graphic data compresses badly unlike text.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.12.2.4 DefaultValues Property

Used to request default values/expressions from the server and assign them to the DefaultExpression property.

**Class**

**TDADataSetOptions**

**Syntax**

**property** DefaultValues: boolean **default** False;
Remarks

If True, the default values(expressions) are requested from the server and assigned to the DefaultExpression property of TField objects replacing already existent values.

5.11.1.12.2.5 DetailDelay Property

Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.

Class

TDADataSetOptions

Syntax

| property | DetailDelay: integer default 0; |

Remarks

Use the DetailDelay property to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset. If DetailDelay is 0 (the default value) then refreshing of detail dataset occurs immediately. The DetailDelay option should be used for detail dataset.

5.11.1.12.2.6 FieldsOrigin Property

Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.

Class

TDADataSetOptions

Syntax

| property | FieldsOrigin: boolean; |

Remarks
If True, TCustomDADataSet fills the Origin property of the TField objects by appropriate value when opening a dataset.

### 5.11.1.12.2.7 FlatBuffers Property

Used to control how a dataset treats data of the ftString and ftVarBytes fields.

**Class**

`TDADatasetOptions`

**Syntax**

```plaintext
property FlatBuffers: boolean default False;
```

**Remarks**

Use the FlatBuffers property to control how a dataset treats data of the ftString and ftVarBytes fields. When set to True, all data fetched from the server is stored in record pdata without unused tails.

### 5.11.1.12.2.8 InsertAllSetFields Property

Used to include all set dataset fields in the generated INSERT statement.

**Class**

`TDADatasetOptions`

**Syntax**

```plaintext
property InsertAllSetFields: boolean default False;
```

**Remarks**

If True, all set dataset fields, including those set to NULL explicitly, will be included in the generated INSERT statements. Otherwise, only set fields containing not NULL values will be included to the generated INSERT statement.
5.11.1.12.2.9 LocalMasterDetail Property

Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.

Class
TDADatasetOptions

Syntax

```
property LocalMasterDetail: boolean default False;
```

Remarks

If True, for detail dataset in master-detail relationship TCustomDADataSet uses local filtering for establishing master/detail relationship and does not refer to the server. Otherwise detail dataset performs query each time a record is selected in master dataset. This option is useful for reducing server calls number, server resources economy. It can be useful for slow connection. The TMemDataSet.CachedUpdates mode can be used for detail dataset only when this option is set to true. Setting the LocalMasterDetail option to True is not recommended when detail table contains too many rows, because when it is set to False, only records that correspond to the current record in master dataset are fetched.

5.11.1.12.2.10 LongStrings Property

Used to represent string fields with the length that is greater than 255 as TStringField.

Class
TDADatasetOptions

Syntax

```
property LongStrings: boolean default True;
```

Remarks

Use the LongStrings property to represent string fields with the length that is greater than 255.
as TStringField, not as TMemoField.

5.11.1.12.2.11 MasterFieldsNullable Property

Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).

Class

TDADatasetOptions

Syntax

property MasterFieldsNullable: boolean default False;

5.11.1.12.2.12 NumberRange Property

Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Class

TDADatasetOptions

Syntax

property NumberRange: boolean default False;

Remarks

Use the NumberRange property to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
5.11.1.12.2.13  QueryRecCount Property

Used for TCustomDADataset to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.

Class
TDADatasetOptions

Syntax

```property QueryRecCount: boolean default False;```

Remarks
If True, and the FetchAll property is False, TCustomDADataset performs additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records. Does not have any effect if the FetchAll property is True.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.12.2.14  QuoteNames Property

Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.

Class
TDADatasetOptions

Syntax

```property QuoteNames: boolean default False;```

Remarks
If True, TCustomDADataset quotes all database object names in autogenerated SQL statements such as update SQL.
5.11.1.12.2.15  RemoveOnRefresh Property

Used for a dataset to locally remove a record that can not be found on the server.

Class

TDADatasetOptions

Syntax

property RemoveOnRefresh: boolean default True;

Remarks

When the RefreshRecord procedure can't find necessary record on the server and RemoveOnRefresh is set to True, dataset removes the record locally. Usually RefreshRecord can't find necessary record when someone else dropped the record or changed the key value of it.

This option makes sense only if the StrictUpdate option is set to False. If the StrictUpdate option is True, error will be generated regardless of the RemoveOnRefresh option value.

5.11.1.12.2.16  RequiredFields Property

Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.

Class

TDADatasetOptions

Syntax

property RequiredFields: boolean default True;

Remarks

If True, TCustomDADataSet sets the Required property of the TField objects for the NOT NULL fields. It is useful when table has a trigger which updates the NOT NULL fields.
5.11.1.12.2.17  ReturnParams Property

Used to return the new value of fields to dataset after insert or update.

Class

TDADatasetOptions

Syntax

```
property ReturnParams: boolean default False;
```

Remarks

Use the ReturnParams property to return the new value of fields to dataset after insert or update. The actual value of field after insert or update may be different from the value stored in the local memory if the table has a trigger. When ReturnParams is True, OUT parameters of the SQLInsert and SQLUpdate statements is assigned to the corresponding fields.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.12.2.18  SetFieldsReadOnly Property

Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.

Class

TDADatasetOptions

Syntax

```
property SetFieldsReadOnly: boolean default True;
```

Remarks

If True, dataset sets the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated. Set this option for datasets that use automatic generation of the update SQL statements only.
5.11.1.12.2.19  StrictUpdate Property

Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.

Class
TDADatasetOptions

Syntax

```delphi
property StrictUpdate: boolean default True;
```

Remarks

If True, TCustomDADataset raises an exception when the number of updated or deleted records is not equal 1. Setting this option also causes the exception if the RefreshRecord procedure returns more than one record. The exception does not occur when you execute SQL query, that doesn't return resultset.

**Note:** There can be problems if this option is set to True and triggers for UPDATE, DELETE, REFRESH commands that are defined for the table. So it is recommended to disable (set to False) this option with triggers.

TrimFixedChar specifies whether to discard all trailing spaces in the string fields of a dataset.

5.11.1.12.2.20  TrimFixedChar Property

Specifies whether to discard all trailing spaces in the string fields of a dataset.

Class
TDADatasetOptions

Syntax

```delphi
property TrimFixedChar: boolean default True;
```

Remarks

Specifies whether to discard all trailing spaces in the string fields of a dataset.
5.11.1.12.2.21 UpdateAllFields Property

Used to include all dataset fields in the generated UPDATE and INSERT statements.

Class

TDADatasetOptions

Syntax

```plaintext
property UpdateAllFields: boolean default False;
```

Remarks

If True, all dataset fields will be included in the generated UPDATE and INSERT statements. Unspecified fields will have NULL value in the INSERT statements. Otherwise, only updated fields will be included to the generated update statements.

5.11.1.12.2.22 UpdateBatchSize Property

Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Class

TDADatasetOptions

Syntax

```plaintext
property UpdateBatchSize: Integer default 1;
```

Remarks

Use the UpdateBatchSize property to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch. Takes effect only when updating dataset in the TMemDataSet.CachedUpdates mode. The default value is 1.
5.11.1.13 TDAEncryption Class

Used to specify the options of the data encryption in a dataset.

For a list of all members of this type, see TDAEncryption members.

Unit

DBAccess

Syntax

TDAEncryption = class(TPersistent);

Remarks

Set the properties of Encryption to specify the options of the data encryption in a dataset.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryptor</td>
<td>Used to specify the encryptor class that will perform the data encryption.</td>
</tr>
<tr>
<td>Fields</td>
<td>Used to set field names for which encryption will be performed.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback

5.11.1.13.1 Members

TDAEncryption class overview.

5.11.1.13.2 Properties

Properties of the TDAEncryption class.

For a complete list of the TDAEncryption class members, see the TDAEncryption Members topic.
Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryptor</td>
<td>Used to specify the encryptor class that will perform the data encryption.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields</td>
<td>Used to set field names for which encryption will be performed.</td>
</tr>
</tbody>
</table>

See Also
- [TDAEncryption Class](#)
- [TDAEncryption Class Members](#)

5.11.1.13.2.1 Encryptor Property

Used to specify the encryptor class that will perform the data encryption.

Class

[TDAEncryption](#)

Syntax

```plaintext
property Encryptor: TCREncryptor;
```

Remarks

Use the Encryptor property to specify the encryptor class that will perform the data encryption.

© 1997-2021
Devart. All Rights Reserved.

Request Support | DAC Forum | Provide Feedback
5.11.1.13.2.2 Fields Property

Used to set field names for which encryption will be performed.

Class

TDAEncryption

Syntax

```property Fields: string;```

Remarks

Used to set field names for which encryption will be performed. Field names must be separated by semicolons.

5.11.1.14 TDAMapRule Class

Class that forms rules for Data Type Mapping.

For a list of all members of this type, see TDAMapRule members.

Unit

DBAccess

Syntax

```TDAMapRule = class(TMapRule);```

Remarks

Using properties of this class, it is possible to change parameter values of the specified rules from the TDAMapRules set.

Inheritance Hierarchy

TMapRule
  TDAMapRule
5.11.1.14.1 Members

**TDAMapRule** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLengthMax</td>
<td>Maximum DB field length, until which the rule is applied.</td>
</tr>
<tr>
<td>DBLengthMin</td>
<td>Minimum DB field length, starting from which the rule is applied.</td>
</tr>
<tr>
<td>DBScaleMax</td>
<td>Maximum DB field scale, until which the rule is applied to the specified DB field.</td>
</tr>
<tr>
<td>DBScaleMin</td>
<td>Minimum DB field Scale, starting from which the rule is applied to the specified DB field.</td>
</tr>
<tr>
<td>DBType</td>
<td>DB field type, that the rule is applied to.</td>
</tr>
<tr>
<td>FieldLength</td>
<td>The resultant field length in Delphi.</td>
</tr>
<tr>
<td>FieldName</td>
<td>DataSet field name, for which the rule is applied.</td>
</tr>
<tr>
<td>FieldScale</td>
<td>The resultant field Scale in Delphi.</td>
</tr>
<tr>
<td>FieldType</td>
<td>Delphi field type, that the specified DB type or DataSet field will be mapped to.</td>
</tr>
<tr>
<td>IgnoreErrors</td>
<td>Ignoring errors when converting data from DB to Delphi type.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

5.11.1.14.2 Properties

Properties of the **TDAMapRule** class.

For a complete list of the **TDAMapRule** class members, see the **TDAMapRule Members**.
Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLengthMax</td>
<td>Maximum DB field length, until which the rule is applied.</td>
</tr>
<tr>
<td>DBLengthMin</td>
<td>Minimum DB field length, starting from which the rule is applied.</td>
</tr>
<tr>
<td>DBScaleMax</td>
<td>Maximum DB field scale, until which the rule is applied to the specified DB field.</td>
</tr>
<tr>
<td>DBScaleMin</td>
<td>Minimum DB field Scale, starting from which the rule is applied to the specified DB field.</td>
</tr>
<tr>
<td>DBType</td>
<td>DB field type, that the rule is applied to.</td>
</tr>
<tr>
<td>FieldLength</td>
<td>The resultant field length in Delphi.</td>
</tr>
<tr>
<td>FieldName</td>
<td>DataSet field name, for which the rule is applied.</td>
</tr>
<tr>
<td>FieldScale</td>
<td>The resultant field Scale in Delphi.</td>
</tr>
<tr>
<td>FieldType</td>
<td>Delphi field type, that the specified DB type or DataSet field will be mapped to.</td>
</tr>
<tr>
<td>IgnoreErrors</td>
<td>Ignoring errors when converting data from DB to Delphi type.</td>
</tr>
</tbody>
</table>

See Also
- TDAMapRule Class
- TDAMapRule Class Members

© 1997-2021 Devart. All Rights Reserved.
5.11.1.14.2.1  DBLengthMax Property

Maximum DB field length, until which the rule is applied.

Class
TDAMapRule

Syntax

```pascal
property DBLengthMax: Integer default rlAny;
```

Remarks
Setting maximum DB field length, until which the rule is applied to the specified DB field.

5.11.1.14.2.2  DBLengthMin Property

Minimum DB field length, starting from which the rule is applied.

Class
TDAMapRule

Syntax

```pascal
property DBLengthMin: Integer default rlAny;
```

Remarks
Setting minimum DB field length, starting from which the rule is applied to the specified DB field.

5.11.1.14.2.3  DBScaleMax Property

Maximum DB field scale, until which the rule is applied to the specified DB field.

Class
TDAMapRule
5.11.14.2.4 DBScaleMax Property

Setting maximum DB field scale, until which the rule is applied to the specified DB field.

Class
TDAMapRule

Syntax

```property DBScaleMax: Integer default rlAny;```

Remarks

Setting maximum DB field scale, until which the rule is applied to the specified DB field.

5.11.14.2.5 DBScaleMin Property

Minimum DB field Scale, starting from which the rule is applied to the specified DB field.

Class
TDAMapRule

Syntax

```property DBScaleMin: Integer default rlAny;```

Remarks

Setting minimum DB field Scale, starting from which the rule is applied to the specified DB field.

5.11.14.2.5 DbType Property

DB field type, that the rule is applied to.

Class
TDAMapRule

Syntax

```property DbType: Word default dtUnknown;```

Remarks

Setting DB field type, that the rule is applied to. If the current rule is set for Connection, the
The resultant field length in Delphi.

Class

TDAMapRule

Syntax

```property FieldLength: Integer default rlAny;```  

Remarks

Setting the Delphi field length after conversion.

5.11.14.2.7 FieldName Property

DataSet field name, for which the rule is applied.

Class

TDAMapRule

Syntax

```property FieldName: string;```  

Remarks

Specifies the DataSet field name, that the rule is applied to. If the current rule is set for Connection, the rule will be applied to all fields with such name in DataSets related to this Connection.
5.11.1.14.2.8 FieldScale Property

The resultant field Scale in Delphi.

Class
TDAMapRule

Syntax

```plaintext
property FieldScale: Integer default rlAny;
```

Remarks
Setting the Delphi field Scale after conversion.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.14.2.9 FieldType Property

Delphi field type, that the specified DB type or DataSet field will be mapped to.

Class
TDAMapRule

Syntax

```plaintext
property FieldType: TFieldType stored IsFieldTypeStored default ftUnknown;
```

Remarks
Setting Delphi field type, that the specified DB type or DataSet field will be mapped to.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.14.2.10 IgnoreErrors Property

Ignoring errors when converting data from DB to Delphi type.

Class
TDAMapRule
Syntax

```property`` IgnoreErrors: Boolean ``default`` False;
```

Remarks

Allows to ignore errors while data conversion in case if data or DB data format cannot be recorded to the specified Delphi field type. The default value is false.

5.11.15 TDAMapRules Class

Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.

For a list of all members of this type, see `TDAMapRules` members.

Unit

`DBAccess`

Syntax

```TDAMapRules = class(TMapRules);```

Inheritance Hierarchy

TMapRules

TDAMapRules

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgnoreInvalidRules</td>
<td>Used to avoid raising exception on mapping rules that can't be applied.</td>
</tr>
</tbody>
</table>
Properties of the **TDAMapRules** class.

For a complete list of the **TDAMapRules** class members, see the [TDAMapRules Members](#) topic.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgnoreInvalidRules</td>
<td>Used to avoid raising exception on mapping rules that can't be applied.</td>
</tr>
</tbody>
</table>

### See Also
- [TDAMapRules Class](#)
- [TDAMapRules Class Members](#)

#### IgnoreInvalidRules Property

Used to avoid raising exception on mapping rules that can't be applied.

**Class**

[TDAMapRules](#)

**Syntax**

```objectivec
property IgnoreInvalidRules: boolean default False;
```

**Remarks**

Allows to ignore errors (not to raise exception) during data conversion in case if the data or DB data format cannot be recorded to the specified Delphi field type. The default value is false.

**Note:** In order to ignore errors occurring during data conversion, use the
TDAMapRule.IgnoreErrors property

See Also

• TDAMapRule.IgnoreErrors

5.11.1.16 TDAMetaData Class

A class for retrieving metainformation of the specified database objects in the form of dataset.

For a list of all members of this type, see TDAMetaData members.

Unit

DBAccess

Syntax

TDAMetaData = class(TMemDataSet);

Remarks

TDAMetaData is a TDataSet descendant standing for retrieving metainformation of the specified database objects in the form of dataset. First of all you need to specify which kind of metainformation you want to see. For this you need to assign the TDAMetaData.MetaDataKind property. Provide one or more conditions in the TDAMetaData.Restrictions property to diminish the size of the resultset and get only information you are interested in.

Use the TDAMetaData.GetMetaDataKinds method to get the full list of supported kinds of meta data. With the TDAMetaData.GetRestrictions method you can find out what restrictions are applicable to the specified MetaDataKind.

Example

The code below demonstrates how to get information about columns of the 'emp' table:

```plaintext
MetaData.Connection := Connection;
MetaData.MetaDataKind := 'Columns';
MetaData.Restrictions.Values['TABLE_NAME'] := 'Emp';
MetaData.Open;
```

Inheritance Hierarchy
**TDAMetaData** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MetaDataKind</strong></td>
<td>Used to specify which kind of metainformation to show.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Determines whether a query is prepared for execution or</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
Ranged (inherited from `TMemDataSet`) | Indicates whether a range is applied to a dataset.
---|---
Restrictions | Used to provide one or more conditions restricting the list of objects to be described.
UpdateRecordTypes (inherited from `TMemDataSet`) | Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from `TMemDataSet`) | Used to check the status of the cached updates buffer.

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Retrieves <code>TBlob</code> object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetMetaDataKinds</td>
<td>Used to get values acceptable in the MetaDataKind property.</td>
</tr>
<tr>
<td>GetRestrictions</td>
<td>Used to find out which restrictions are applicable to a certain MetaDataKind.</td>
</tr>
<tr>
<td>Locate (inherited from TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx (inherited from TMemDataSet)</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Prepare (inherited from TMemDataSet)</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td>RestoreUpdates (inherited from TMemDataSet)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord (inherited from TMemDataSet)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveToXML (inherited from TMemDataSet)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange (inherited from TMemDataSet)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>UnPrepare (inherited from TMemDataSet)</td>
<td>Frees the resources allocated for a previously</td>
</tr>
</tbody>
</table>
Properties of the TDAMetaData class.

For a complete list of the TDAMetaData class members, see the TDAMetaData Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>Used to get or set the list of</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnUpdateError</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MetaDataKind</strong></td>
<td>Used to specify which kind of metainformation to show.</td>
</tr>
<tr>
<td><strong>Prepared</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>Used to provide one or more conditions restricting the list of objects to be described.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

**See Also**
- `TDAMetaData Class`
- `TDAMetaData Class Members`

© 1997-2021 Devart. All Rights Reserved.

5.11.1.16.2.1 Connection Property

Used to specify a connection object to use to connect to a data store.

**Class**
- `TDAMetaData`
Syntax

**property** Connection: **TCustomDAConnection**;

Remarks

Use the Connection property to specify a connection object to use to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, set the Connection property to reference an instanciated TCustomDAConnection object.

5.11.1.16.2.2 MetaDataKind Property

Used to specify which kind of metainformation to show.

Class

**TDAMetaData**

Syntax

**property** MetaDataKind: **string**;

Remarks

This string property specifies which kind of metainformation to show. The value of this property should be assigned before activating the component. If MetaDataKind equals to an empty string (the default value), the full value list that this property accepts will be shown.

They are described in the table below:

<table>
<thead>
<tr>
<th>MetaDataKind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>show metainformation about columns of existing tables</td>
</tr>
<tr>
<td>Constraints</td>
<td>show metainformation about the constraints defined in the database</td>
</tr>
<tr>
<td>IndexColumns</td>
<td>show metainformation about indexed columns</td>
</tr>
<tr>
<td>Indexes</td>
<td>show metainformation about indexes in a database</td>
</tr>
<tr>
<td>MetaDataKinds</td>
<td>show the acceptable values of this property. You will get the same result if the MetaDataKind property is an empty string</td>
</tr>
<tr>
<td>ProcedurePara</td>
<td>show metainformation about parameters of existing procedures</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>Procedures</td>
<td>show metainformation about existing procedures</td>
</tr>
<tr>
<td>Restrictions</td>
<td>generates a dataset that describes which restrictions are applicable to each MetaDataKind</td>
</tr>
<tr>
<td>Tables</td>
<td>show metainformation about existing tables</td>
</tr>
<tr>
<td>Databases</td>
<td>show metainformation about existing databases</td>
</tr>
</tbody>
</table>

If you provide a value that equals neither of the values described in the table, an error will be raised.

**See Also**
- Restrictions

5.11.1.16.2.3 Restrictions Property

Used to provide one or more conditions restricting the list of objects to be described.

**Class**

**TDAMetaData**

**Syntax**

```
property Restrictions: TStrings;
```

**Remarks**

Use the Restriction list to provide one or more conditions restricting the list of objects to be described. To see the full list of restrictions and to which metadata kinds they are applicable, you should assign the Restrictions value to the MetaDataKind property and view the result.

**See Also**
- MetaDataKind

© 1997-2021 Devart. All Rights Reserved.
5.11.1.16.3 Methods

Methods of the TDAMetaData class.

For a complete list of the TDAMetaData class members, see the TDAMetaData Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong></td>
<td>(inherited from TMemDataSet) Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>(inherited from TMemDataSet) Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>(inherited from TMemDataSet) Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>(inherited from TMemDataSet) Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>(inherited from TMemDataSet) Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>(inherited from TMemDataSet) Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>(inherited from TMemDataSet) Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>(inherited from TMemDataSet) Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>(inherited from TMemDataSet) Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>GetMetaDataKinds</strong></td>
<td>Used to get values acceptable in the MetaDataKind property.</td>
</tr>
<tr>
<td><strong>GetRestrictions</strong></td>
<td>Used to find out which restrictions are applicable to a certain MetaDataKind.</td>
</tr>
<tr>
<td><strong>Locate</strong></td>
<td>(inherited from TMemDataSet) Overloaded. Searches a dataset for a specific record</td>
</tr>
<tr>
<td>Procedure (inherited from TMemDataSet)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
</tbody>
</table>
**UpdateStatus** (inherited from **TMemDataSet**)

Indicates the current update status for the dataset when cached updates are enabled.

See Also
- **TDAMetaData Class**
- **TDAMetaData Class Members**

5.11.1.16.3.1 GetMetaDataKinds Method

Used to get values acceptable in the MetaDataKind property.

Class

**TDAMetaData**

Syntax

```pascal
procedure GetMetaDataKinds(List: TStrings);
```

Parameters

- **List**
  - Holds the object that will be filled with metadata kinds (restrictions).

Remarks

Call the GetMetaDataKinds method to get values acceptable in the MetaDataKind property. The List parameter will be cleared and then filled with values.

See Also
- **MetaDataKind**
5.11.1.16.3.2 GetRestrictions Method

Used to find out which restrictions are applicable to a certain MetaDataKind.

Class

TDAMetaData

Syntax

```pascal
procedure GetRestrictions(List: TStrings; const MetaDataKind: string);
```

Parameters

- **List**
  - Holds the object that will be filled with metadata kinds (restrictions).
- **MetaDataKind**
  - Holds the metadata kind for which restrictions are returned.

Remarks

Call the GetRestrictions method to find out which restrictions are applicable to a certain MetaDataKind. The List parameter will be cleared and then filled with values.

See Also

- [Restrictions](#)
- [GetMetaDataKinds](#)

5.11.1.17 TDAParam Class

A class that forms objects to represent the values of the parameters set.

For a list of all members of this type, see [TDAParam](#) members.

Unit

DBAccess

Syntax

```pascal
TDAParam = class(TParam);
```
Remarks

Use the properties of TDAParam to set the value of a parameter. Objects that use parameters create TDAParam objects to represent these parameters. For example, TDAParam objects are used by TCustomDASQL, TCustomDADataSet.

TDAParam shares many properties with TField, as both describe the value of a field in a dataset. However, a TField object has several properties to describe the field binding and the way the field is displayed, edited, or calculated, that are not needed in a TDAParam object. Conversely, TDAParam includes properties that indicate how the field value is passed as a parameter.

See Also

- TCustomDADataSet
- TCustomDASQL
- TDAParams

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsBlob</td>
<td>Used to set and read the value of the BLOB parameter as string.</td>
</tr>
<tr>
<td>AsBlobRef</td>
<td>Used to set and read the value of the BLOB parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Used to assign the value for a float field to a parameter.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Used to assign the value for an integer field to the parameter.</td>
</tr>
<tr>
<td>AsLargeInt</td>
<td>Used to assign the value for a LargeInteger field to the parameter.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AsMemo</td>
<td>Used to assign the value for a memo field to the parameter.</td>
</tr>
<tr>
<td>AsMemoRef</td>
<td>Used to set and read the value of the memo parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsSQLTimeStamp</td>
<td>Used to specify the value of the parameter when it represents a SQL timestamp field.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to assign the string value to the parameter.</td>
</tr>
<tr>
<td>AsWideString</td>
<td>Used to assign the Unicode string value to the parameter.</td>
</tr>
<tr>
<td>DataType</td>
<td>Indicates the data type of the parameter.</td>
</tr>
<tr>
<td>IsNull</td>
<td>Used to indicate whether the value assigned to a parameter is NULL.</td>
</tr>
<tr>
<td>ParamType</td>
<td>Used to indicate the type of use for a parameter.</td>
</tr>
<tr>
<td>Size</td>
<td>Specifies the size of a string type parameter.</td>
</tr>
<tr>
<td>Value</td>
<td>Used to represent the value of the parameter as Variant.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignField</td>
<td>Assigns field name and field value to a param.</td>
</tr>
<tr>
<td>AssignFieldValue</td>
<td>Assigns the specified field properties and value to a parameter.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Places the content of a specified file into a TDAParam object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Places the content from a stream into a TDAParam object.</td>
</tr>
<tr>
<td>SetBlobData</td>
<td>Overloaded. Writes the data from a specified buffer to a TDBlob.</td>
</tr>
</tbody>
</table>
Properties of the **TDAParam** class.

For a complete list of the **TDAParam** class members, see the [TDAParam Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsBlob</td>
<td>Used to set and read the value of the BLOB parameter as string.</td>
</tr>
<tr>
<td>AsBlobRef</td>
<td>Used to set and read the value of the BLOB parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Used to assign the value for a float field to a parameter.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Used to assign the value for an integer field to the parameter.</td>
</tr>
<tr>
<td>AsLargeInt</td>
<td>Used to assign the value for a LargeInteger field to the parameter.</td>
</tr>
<tr>
<td>AsMemo</td>
<td>Used to assign the value for a memo field to the parameter.</td>
</tr>
<tr>
<td>AsMemoRef</td>
<td>Used to set and read the value of the memo parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsSQLTimeStamp</td>
<td>Used to specify the value of the parameter when it represents a SQL timestamp field.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to assign the string value to the parameter.</td>
</tr>
<tr>
<td>AsWideString</td>
<td>Used to assign the Unicode string value to the</td>
</tr>
</tbody>
</table>
IsNull

Used to indicate whether the value assigned to a parameter is NULL.

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataType</td>
<td>Indicates the data type of the parameter.</td>
</tr>
<tr>
<td>ParamType</td>
<td>Used to indicate the type of use for a parameter.</td>
</tr>
<tr>
<td>Size</td>
<td>Specifies the size of a string type parameter.</td>
</tr>
<tr>
<td>Value</td>
<td>Used to represent the value of the parameter as Variant.</td>
</tr>
</tbody>
</table>

See Also

- TDAParam Class
- TDAParam Class Members

5.11.1.17.2.1 AsBlob Property

Used to set and read the value of the BLOB parameter as string.

Class

TDAParam

Syntax

```property AsBlob: TBlobData;```

Remarks

Use the AsBlob property to set and read the value of the BLOB parameter as string. Setting AsBlob will set the DataType property to ftBlob.
5.11.1.17.2.2  AsBlobRef Property

Used to set and read the value of the BLOB parameter as a TBlob object.

Class

TDAParam

Syntax

property AsBlobRef: TBlob;

Remarks

Use the AsBlobRef property to set and read the value of the BLOB parameter as a TBlob object. Setting AsBlobRef will set the DataType property to ftBlob.

5.11.1.17.2.3  AsFloat Property

Used to assign the value for a float field to a parameter.

Class

TDAParam

Syntax

property AsFloat: double;

Remarks

Use the AsFloat property to assign the value for a float field to the parameter. Setting AsFloat will set the DataType property to dtFloat.

Read the AsFloat property to determine the value that was assigned to an output parameter, represented as Double. The value of the parameter will be converted to the Double value if possible.
5.11.1.17.2.4  AsInteger Property

Used to assign the value for an integer field to the parameter.

Class

TDAParam

Syntax

```pascal
property AsInteger: LongInt;
```

Remarks

Use the AsInteger property to assign the value for an integer field to the parameter. Setting AsInteger will set the DataType property to dtInteger.

Read the AsInteger property to determine the value that was assigned to an output parameter, represented as a 32-bit integer. The value of the parameter will be converted to the Integer value if possible.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.17.2.5  AsLargeInt Property

Used to assign the value for a LargeInteger field to the parameter.

Class

TDAParam

Syntax

```pascal
property AsLargeInt: Int64;
```

Remarks

Set the AsLargeInt property to assign the value for an Int64 field to the parameter. Setting AsLargeInt will set the DataType property to dtLargeint.

Read the AsLargeInt property to determine the value that was assigned to an output parameter, represented as a 64-bit integer. The value of the parameter will be converted to the Int64 value if possible.

© 1997-2021 Devart. All Rights Reserved.
5.11.1.17.2.6  AsMemo Property

Used to assign the value for a memo field to the parameter.

Class
TDAParam

Syntax

property AsMemo: string;

Remarks
Use the AsMemo property to assign the value for a memo field to the parameter. Setting AsMemo will set the DataType property to ftMemo.

5.11.1.17.2.7  AsMemoRef Property

Used to set and read the value of the memo parameter as a TBlob object.

Class
TDAParam

Syntax

property AsMemoRef: TBlob;

Remarks
Use the AsMemoRef property to set and read the value of the memo parameter as a TBlob object. Setting AsMemoRef will set the DataType property to ftMemo.
5.11.1.17.2.8 AsSQLTimeStamp Property

Used to specify the value of the parameter when it represents a SQL timestamp field.

Class

TDAParam

Syntax

property AsSQLTimeStamp: TSQLTimeStamp;

Remarks

Set the AsSQLTimeStamp property to assign the value for a SQL timestamp field to the parameter. Setting AsSQLTimeStamp sets the DataType property to ftTimeStamp.

5.11.1.17.2.9 AsString Property

Used to assign the string value to the parameter.

Class

TDAParam

Syntax

property AsString: string;

Remarks

Use the AsString property to assign the string value to the parameter. Setting AsString will set the DataType property to ftString.

Read the AsString property to determine the value that was assigned to an output parameter represented as a string. The value of the parameter will be converted to a string.
5.11.1.17.2.10  AsWideString Property

Used to assign the Unicode string value to the parameter.

Class

TDAParam

Syntax

\texttt{property AsWideString: string; }

Remarks

Set AsWideString to assign the Unicode string value to the parameter. Setting AsWideString will set the DataType property to ftWideString.

Read the AsWideString property to determine the value that was assigned to an output parameter, represented as a Unicode string. The value of the parameter will be converted to a Unicode string.

5.11.1.17.2.11  DataType Property

Indicates the data type of the parameter.

Class

TDAParam

Syntax

\texttt{property DataType: TFieldType stored IsDataTypeStored; }

Remarks

DataType is set automatically when a value is assigned to a parameter. Do not set DataType for bound fields, as this may cause the assigned value to be misinterpreted.

Read DataType to learn the type of data that was assigned to the parameter. Every possible value of DataType corresponds to the type of a database field.
5.11.1.17.2.12  IsNull Property

Used to indicate whether the value assigned to a parameter is NULL.

Class
TDAParam

Syntax

```PropertyValue IsNull: boolean;
```

Remarks
Use the IsNull property to indicate whether the value assigned to a parameter is NULL.

5.11.1.17.2.13  ParamType Property

Used to indicate the type of use for a parameter.

Class
TDAParam

Syntax

```PropertyValue ParamType default DB . ptUnknown;
```

Remarks
Objects that use TDAParam objects to represent field parameters set ParamType to indicate the type of use for a parameter.

To learn the description of TParamType refer to Delphi Help.
**TDAParam**

**Syntax**

```plaintext
property size: integer default 0;
```

**Remarks**

Use the Size property to indicate the maximum number of characters the parameter may contain. Use the Size property only for Output parameters of the `ftString`, `ftFixedChar`, `ftBytes`, `ftVarBytes`, or `ftWideString` type.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.17.2.15 Value Property

Used to represent the value of the parameter as Variant.

**Class**

`TDAParam`

**Syntax**

```plaintext
property value: variant stored IsValueStored;
```

**Remarks**

The Value property represents the value of the parameter as Variant.

Use Value in generic code that manipulates the values of parameters without the need to know the field type the parameter represent.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.17.3 Methods

Methods of the `TDAParam` class.

For a complete list of the `TDAParam` class members, see the `TDAParam Members` topic.

© 2021 Devart
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignField</td>
<td>Assigns field name and field value to a param.</td>
</tr>
<tr>
<td>AssignFieldValue</td>
<td>Assigns the specified field properties and value to a parameter.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Places the content of a specified file into a TDAParam object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Places the content from a stream into a TDAParam object.</td>
</tr>
<tr>
<td>SetBlobData</td>
<td>Overloaded. Writes the data from a specified buffer to BLOB.</td>
</tr>
</tbody>
</table>

See Also
- TDAParam Class
- TDAParam Class Members

5.11.1.17.3.1 AssignField Method

Assigns field name and field value to a param.

Class
- TDAParam

Syntax

```
procedure AssignField(Field: TField);
```

Parameters

- `Field`
  
  Holds the field which name and value should be assigned to the param.

Remarks

Call the AssignField method to assign field name and field value to a param.
### 5.11.1.17.3.2 AssignFieldValue Method

Assigns the specified field properties and value to a parameter.

**Class**

**TDAParam**

**Syntax**

```delphi
procedure AssignFieldValue(Field: TField; const Value: Variant);
virtual;
```

**Parameters**

- **Field**
  - Holds the field the properties of which will be assigned to the parameter.
- **Value**
  - Holds the value for the parameter.

**Remarks**

Call the AssignFieldValue method to assign the specified field properties and value to a parameter.

© 1997-2021 Devart. All Rights Reserved.

### 5.11.1.17.3.3 LoadFromFile Method

Places the content of a specified file into a TDAParam object.

**Class**

**TDAParam**

**Syntax**

```delphi
procedure LoadFromFile(const FileName: string; BlobType: TBlobType);
```

**Parameters**

- **FileName**
  - Holds the name of the file.
- **BlobType**
Holds a value that modifies the DataType property so that this TDAParam object now holds the BLOB value.

**Remarks**

Use the LoadFromFile method to place the content of a file specified by FileName into a TDAParam object. The BlobType value modifies the DataType property so that this TDAParam object now holds the BLOB value.

**See Also**

- LoadFromStream
5.11.1.17.3.5 SetBlobData Method

Writes the data from a specified buffer to BLOB.

Class

TDAParam

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetBlobData(Buffer: TValueBuffer)</td>
<td>Writes the data from a specified buffer to BLOB.</td>
</tr>
<tr>
<td>SetBlobData(Buffer: IntPtr; Size: Integer)</td>
<td>Writes the data from a specified buffer to BLOB.</td>
</tr>
</tbody>
</table>

Writes the data from a specified buffer to BLOB.

Class

TDAParam

Syntax

```
procedure SetBlobData(Buffer: TValueBuffer); overload;
```

Parameters

Buffer

Holds the pointer to the data.
### Syntax

```plaintext
procedure SetBlobData(Buffer: IntPtr; Size: Integer); overload;
```

### Parameters

- **Buffer**
  - Holds the pointer to data.
- **Size**
  - Holds the number of bytes to read from the buffer.

### Remarks

Call the `SetBlobData` method to write data from a specified buffer to BLOB.

5.11.1.18 TDAParams Class

This class is used to manage a list of TDAParam objects for an object that uses field parameters.

For a list of all members of this type, see `TDAParams` members.

### Unit

`DBAccess`

### Syntax

```plaintext
TDAParams = class(TParams);
```

### Remarks

Use `TDAParams` to manage a list of TDAParam objects for an object that uses field parameters. For example, `TCustomDADataSet` objects and `TCustomDASQL` objects use `TDAParams` objects to create and access their parameters.

### See Also

- `TCustomDADataSet.Params`
- `TCustomDASQL.Params`
- `TDAParam`
5.11.1.18.1  Members

**TDAParams** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all parameters.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindParam</td>
<td>Searches for a parameter with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Searches for a parameter with the specified name.</td>
</tr>
</tbody>
</table>

5.11.1.18.2  Properties

Properties of the **TDAParams** class.

For a complete list of the **TDAParams** class members, see the [TDAParams Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all parameters.</td>
</tr>
</tbody>
</table>

See Also

- [TDAParams Class](#)
- [TDAParams Class Members](#)
5.11.1.18.2.1 Items Property (Indexer)

Used to iterate through all parameters.

Class

TDAParams

Syntax

```
property Items[Index: integer]: TDAParam; default;
```

Parameters

*Index*

Holds an index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all parameters. Index identifies the index in the range 0..Count - 1. Items can reference a particular parameter by its index, but the ParamByName method is preferred in order to avoid depending on the order of the parameters.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.18.3 Methods

Methods of the TDAParams class.

For a complete list of the TDAParams class members, see the TDAParams Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindParam</td>
<td>Searches for a parameter with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Searches for a parameter with the specified name.</td>
</tr>
</tbody>
</table>

See Also

- TDAParams Class
- TDAParams Class Members

© 1997-2021 Devart. All Rights Reserved.
5.11.1.18.3.1  FindParam Method

Searches for a parameter with the specified name.

Class
TDAParams

Syntax
function FindParam(const Value: string): TDAParam;

Parameters
Value
  Holds the parameter name.

Return Value
  a parameter, if a match was found. Nil otherwise.

Remarks
Use the FindParam method to find a parameter with the name passed in Value. If a match is found, FindParam returns the parameter. Otherwise, it returns nil. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

To locate more than one parameter at a time by name, use the GetParamList method instead. To get only the value of a named parameter, use the ParamValues property.

5.11.1.18.3.2  ParamByName Method

Searches for a parameter with the specified name.

Class
TDAParams

Syntax
function ParamByName(const Value: string): TDAParam;

Parameters
Value
Holds the parameter name.

**Return Value**

A parameter, if the match was found. Otherwise an exception is raised.

**Remarks**

Use the `ParamByName` method to find a parameter with the name passed in `Value`. If a match was found, `ParamByName` returns the parameter. Otherwise, an exception is raised. Use this method rather than a direct reference to the `Items` property to avoid depending on the order of the entries.

To locate a parameter by name without raising an exception if the parameter is not found, use the `FindParam` method.

---

**5.11.1.19 TDATransaction Class**

A base class that implements functionality for controlling transactions.

For a list of all members of this type, see the `TDATransaction` members.

**Unit**

`DBAccess`

**Syntax**

```
TDATransaction = class(TComponent);
```

**Remarks**

`TDATransaction` is a base class for components implementing functionality for managing transactions.

Do not create instances of `TDATransaction`. Use descendants of the `TDATransaction` class instead.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if the transaction is active.</td>
</tr>
<tr>
<td>DefaultCloseAction</td>
<td>Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>Commits the current transaction.</td>
</tr>
<tr>
<td>Rollback</td>
<td>Discards all modifications of data associated with the current transaction and ends the transaction.</td>
</tr>
<tr>
<td>StartTransaction</td>
<td>Begins a new transaction.</td>
</tr>
</tbody>
</table>

## Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnCommit</td>
<td>Occurs after the transaction has been successfully committed.</td>
</tr>
<tr>
<td>OnCommitRetaining</td>
<td>Occurs after CommitRetaining has been executed.</td>
</tr>
<tr>
<td>OnError</td>
<td>Used to process errors that occur during executing a transaction.</td>
</tr>
<tr>
<td>OnRollback</td>
<td>Occurs after the transaction has been successfully rolled back.</td>
</tr>
<tr>
<td>OnRollbackRetaining</td>
<td>Occurs after RollbackRetaining has been executed.</td>
</tr>
</tbody>
</table>
Properties of the TDATransaction class.

For a complete list of the TDATransaction class members, see the TDATransaction Members topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if the transaction is active.</td>
</tr>
<tr>
<td>DefaultCloseAction</td>
<td>Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.</td>
</tr>
</tbody>
</table>

**See Also**
- TDATransaction Class
- TDATransaction Class Members

**Remarks**

Indicates whether the transaction is active. This property is read-only.
5.11.1.19.2.2 DefaultCloseAction Property

Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Class

**TDATransaction**

Syntax

```
property DefaultCloseAction: TCRTransactionAction default taRollback;
```

Remarks

Use DefaultCloseAction to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

5.11.1.19.3 Methods

Methods of the **TDATransaction** class.

For a complete list of the **TDATransaction** class members, see the **TDATransaction Members** topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>Commits the current transaction.</td>
</tr>
<tr>
<td>Rollback</td>
<td>Discards all modifications of data associated with the current transaction and ends the transaction.</td>
</tr>
<tr>
<td>StartTransaction</td>
<td>Begins a new transaction.</td>
</tr>
</tbody>
</table>
See Also
- **TDATransaction Class**
- **TDATransaction Class Members**

5.11.1.19.3.1 Commit Method

Commits the current transaction.

Class
**TDATransaction**

Syntax

```
procedure Commit; virtual;
```

Remarks
Call the Commit method to commit the current transaction. On commit server writes permanently all pending data updates associated with the current transaction to the database, and then finishes the transaction.

See Also
- **Rollback**
- **StartTransaction**

5.11.1.19.3.2 Rollback Method

Discards all modifications of data associated with the current transaction and ends the transaction.

Class
**TDATransaction**

Syntax
**procedure** Rollback; **virtual**;

Remarks

Call Rollback to cancel all data modifications made within the current transaction to the database server, and finish the transaction.

See Also

- Commit
- StartTransaction

© 1997-2021
Devart. All Rights Reserved.

5.11.1.19.3.3  StartTransaction Method

Begins a new transaction.

Class

**TDATransaction**

Syntax

**procedure** StartTransaction; **virtual**;

Remarks

Call the StartTransaction method to begin a new transaction against the database server. Before calling StartTransaction, an application should check the **Active** property. If TDATransaction.Active is True, indicating that a transaction is already in progress, a subsequent call to StartTransaction will raise EDatabaseError. An active transaction must be finished by call to Commit or Rollback before call to StartTransaction. Call to StartTransaction when connection is closed also will raise EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until the application calls Commit to save the changes, or Rollback to cancel them.

See Also

- Commit
- Rollback

© 1997-2021
Devart. All Rights Reserved.
 Deprecated.

5.11.1.19.4 Events

Events of the **TDATransaction** class.

For a complete list of the **TDATransaction** class members, see the **TDATransaction Members** topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnCommit</td>
<td>Occurs after the transaction has been successfully committed.</td>
</tr>
<tr>
<td>OnCommitRetaining</td>
<td>Occurs after CommitRetaining has been executed.</td>
</tr>
<tr>
<td>OnError</td>
<td>Used to process errors that occur during executing a transaction.</td>
</tr>
<tr>
<td>OnRollback</td>
<td>Occurs after the transaction has been successfully rolled back.</td>
</tr>
<tr>
<td>OnRollbackRetaining</td>
<td>Occurs after RollbackRetaining has been executed.</td>
</tr>
</tbody>
</table>

See Also

- **TDATransaction Class**
- **TDATransaction Class Members**

© 1997-2021
Devar. All Rights Reserved.

5.11.1.19.4.1 OnCommit Event

Occurs after the transaction has been successfully committed.

Class

**TDATransaction**

Syntax
property OnCommit: TNotifyEvent;

Remarks
The OnCommit event fires when the M:Devart.Dac.TDATransaction.Commit method is executed, just after the transaction is successfully committed. In order to respond to the M:Devart.PgDac.TPgTransaction.CommitRetaining() method execution, the OnCommitRetaining event is used. When an error occurs during commit, the OnError event fires.

See Also
- Commit
- OnError

© 1997-2021 Devart. All Rights Reserved.

5.11.1.19.4.2 OnCommitRetaining Event

Occurs after CommitRetaining has been executed.

Class
TDATransaction

Syntax
property OnCommitRetaining: TNotifyEvent;

Remarks
The OnCommitRetaining event fires when the CommitRetaining method is executed, just after the transaction is successfully committed. In order to respond to the M:Devart.Dac.TDATransaction.Commit method execution, the OnCommit event is used.

When an error occurs during commit, the OnError event fired.

See Also
- Commit
- OnCommit
- OnError

© 1997-2021 Devart. All Rights Reserved.
5.11.1.19.4.3 OnError Event

Used to process errors that occur during executing a transaction.

**Class**

`TDATransaction`

**Syntax**

```property OnError: TDATransactionErrorEvent;```

**Remarks**

Add a handler to the OnError event to process errors that occur during executing a transaction control statements such as `Commit`, `Rollback`. Check the E parameter to get the error code.

**See Also**

- `Commit`
- `Rollback`
- `StartTransaction`

5.11.1.19.4.4 OnRollback Event

Occurs after the transaction has been successfully rolled back.

**Class**

`TDATransaction`

**Syntax**

```property OnRollback: TNotifyEvent;```

**Remarks**

The OnRollback event fires when the M:Devart.Dac.TDATransaction.Rollback method is executed, just after the transaction is successfully rolled back. In order to respond to the M:Devart.PgDac.TPgTransaction.RollbackRetaining() method execution, the
OnRollbackRetaining event is used.

When an error occurs during rollback, the **OnError** event fired.

### See Also
- **Rollback**
- **OnError**

### 5.11.1.19.4.5 OnRollbackRetaining Event

Occurs after RollbackRetaining has been executed.

### Class
**TDATransaction**

### Syntax

```pascal
property OnRollbackRetaining: TNotifyEvent;
```

### Remarks

The OnRollbackRetaining event fires when the RollbackRetaining method is executed, just after the transaction is successfully rolled back. In order to respond to the **Rollback** method execution, the **OnRollback** event is used. When an error occurs during rollback, the **OnError** event fired.

### See Also
- **Rollback**
- **OnRollback**
- **OnError**

### 5.11.1.20 TMacro Class

Object that represents the value of a macro.

For a list of all members of this type, see **TMacro** members.
Unit

DBAccess

Syntax

`TMacro = class(TCollectionItem);`

Remarks

TMacro object represents the value of a macro. Macro is a variable that holds string value. You just insert `& MacroName` in a SQL query text and change the value of macro by the Macro property editor at design time or the Value property at run time. At the time of opening query macro is replaced by its value.

If by any reason it is not convenient for you to use the `&` symbol as a character of macro replacement, change the value of the MacroChar variable.

See Also

- TMacros

© 1997-2021 Devart. All Rights Reserved.

5.11.1.20.1 Members

**TMacro** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if the macro should be expanded.</td>
</tr>
<tr>
<td>AsDateTime</td>
<td>Used to set the TDateTime value to a macro.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Used to set the float value to a macro.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Used to set the integer value to a macro.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to assign the string value to a macro.</td>
</tr>
<tr>
<td>Name</td>
<td>Used to identify a particular macro.</td>
</tr>
</tbody>
</table>
Properties of the TMacro class.

For a complete list of the TMacro class members, see the TMacro Members topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsDateTime</td>
<td>Used to set the TDateTime value to a macro.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Used to set the float value to a macro.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Used to set the integer value to a macro.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to assign the string value to a macro.</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if the macro should be expanded.</td>
</tr>
<tr>
<td>Name</td>
<td>Used to identify a particular macro.</td>
</tr>
<tr>
<td>Value</td>
<td>Used to set the value to a macro.</td>
</tr>
</tbody>
</table>

See Also
- TMacro Class
- TMacro Class Members
5.11.1.20.2.1  Active Property

Used to determine if the macro should be expanded.

Class

TMacro

Syntax

| property Active: boolean default True; |

Remarks

When set to True, the macro will be expanded, otherwise macro definition is replaced by null string. You can use the Active property to modify the SQL property.

The default value is True.

Example

PgQuery.SQL.Text := 'SELECT * FROM Dept WHERE DeptNo > 20 &Cond1';
PgQuery.Macros[0].Value := 'and DName is NULL';
PgQuery.Macros[0].Active := False;

5.11.1.20.2.2  AsDateTime Property

Used to set the TDateTime value to a macro.

Class

TMacro

Syntax

| property AsDateTime: TDateTime; |

Remarks

Use the AsDateTime property to set the TDateTime value to a macro.
5.11.1.20.2.3 AsFloat Property

Used to set the float value to a macro.

Class
TMacro

Syntax

```pascal
property AsFloat: double;
```

Remarks
Use the AsFloat property to set the float value to a macro.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.20.2.4 AsInteger Property

Used to set the integer value to a macro.

Class
TMacro

Syntax

```pascal
property AsInteger: integer;
```

Remarks
Use the AsInteger property to set the integer value to a macro.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.20.2.5 AsString Property

Used to assign the string value to a macro.

Class
TMacro
Syntax

```property AsString: string;```

Remarks
Use the AsString property to assign the string value to a macro. Read the AsString property to determine the value of macro represented as a string.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.20.2.6 Name Property

Used to identify a particular macro.

Class

TMacro

Syntax

```property Name: string;```

Remarks
Use the Name property to identify a particular macro.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.20.2.7 Value Property

Used to set the value to a macro.

Class

TMacro

Syntax

```property Value: string;```

Remarks
Use the Value property to set the value to a macro.
5.11.1.21 TMacros Class

Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDADataSet components.

For a list of all members of this type, see TMacros members.

Unit
 DBAccess

Syntax

TMacros = class(TCollection);

Remarks

Use TMacros to manage a list of TMacro objects for the TCustomDASQL or TCustomDADataSet components.

See Also

• TMacro

5.11.1.21.1 Members

TMacros class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all the macros parameters.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignValues</td>
<td>Copies the macros values</td>
</tr>
</tbody>
</table>
### Properties of the **TMacros** class.

For a complete list of the **TMacros** class members, see the [TMacros Members](#) topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Items</strong></td>
<td>Used to iterate through all the macros parameters.</td>
</tr>
</tbody>
</table>

**See Also**

- [TMacros Class](#)
- [TMacros Class Members](#)
Syntax

```pascal
property Items[Index: integer]: TMacro; default;
```

**Parameters**

*Index*

Holds the index in the range 0..Count - 1.

**Remarks**

Use the Items property to iterate through all macros parameters. Index identifies the index in the range 0..Count - 1.

Methods of the **TMacros** class.

For a complete list of the **TMacros** class members, see the **TMacros Members** topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignValues</td>
<td>Copies the macros values and properties from the specified source.</td>
</tr>
<tr>
<td>Expand</td>
<td>Changes the macros in the passed SQL statement to their values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>IsEqual</td>
<td>Compares itself with another TMacro object.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Used to search for a macro with the specified name.</td>
</tr>
<tr>
<td>Scan</td>
<td>Creates a macros from the passed SQL statement.</td>
</tr>
</tbody>
</table>

See Also

- **TMacros Class**
- **TMacros Class Members**
5.11.1.21.3.1 AssignValues Method

Covers the macros values and properties from the specified source.

Class

TMacros

Syntax

```plaintext
procedure AssignValues(Value: TMacros);
```

Parameters

Value

Holds the source to copy the macros values and properties from.

Remarks

The Assign method copies the macros values and properties from the specified source. Macros are not recreated. Only the values of macros with matching names are assigned.

5.11.1.21.3.2 Expand Method

Changes the macros in the passed SQL statement to their values.

Class

TMacros

Syntax

```plaintext
procedure Expand(var SQL: string);
```

Parameters

SQL

Holds the passed SQL statement.

Remarks

Call the Expand method to change the macros in the passed SQL statement to their values.
5.11.1.21.3.3 FindMacro Method

Finds a macro with the specified name.

**Class**

**TMacros**

**Syntax**

```pascal
function FindMacro(const Value: string): TMacro;
```

**Parameters**

- **Value**
  Holds the value of a macro to search for.

**Return Value**

- TMacro object if a match is found, nil otherwise.

**Remarks**

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the `Items` property to avoid depending on the order of the items.

© 1997-2021 Devart. All Rights Reserved.
Return Value
True, if the number of TMacro objects and the values of all TMacro objects are equal.

Remarks
Call the IsEqual method to compare itself with another TMacro object. Returns True if the number of TMacro objects and the values of all TMacro objects are equal.

5.11.1.21.3.5 MacroByName Method

Used to search for a macro with the specified name.

Class
TMacros

Syntax
function MacroByName(const Value: string): TMacro;

Parameters
Value
Holds a name of the macro to search for.

Return Value
TMacro object, if a macro with specified name was found.

Remarks
Call the MacroByName method to find a Macro with the name passed in Value. If a match is found, MacroByName returns the Macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the Items property to avoid depending on the order of the items.

To locate a macro by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.
5.11.1.21.3.6 Scan Method

Creates a macros from the passed SQL statement.

Class

**TMacros**

Syntax

```delphi
procedure Scan(const SQL: string);
```

Parameters

**SQL**

Holds the passed SQL statement.

Remarks

Call the Scan method to create a macros from the passed SQL statement. On that all existing TMacro objects are cleared.

---

5.11.1.22 TPoolingOptions Class

This class allows setting up the behaviour of the connection pool.

For a list of all members of this type, see [TPoolingOptions members](#).

Unit

**DBAccess**

Syntax

```delphi
TPoolingOptions = class(TPersistent);
```

---

5.11.1.22.1 Members

**TPoolingOptions** class overview.

Properties
### Properties of the `TPoolingOptions` class.

For a complete list of the `TPoolingOptions` class members, see the [TPoolingOptions Members](#) topic.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionLifetime</td>
<td>Used to specify the maximum time during which an opened connection can be used by connection pool.</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>Used to specify the maximum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td>MinPoolSize</td>
<td>Used to specify the minimum number of connections that can be opened in the connection pool.</td>
</tr>
<tr>
<td>Validate</td>
<td>Used for a connection to be validated when it is returned from the pool.</td>
</tr>
</tbody>
</table>
ConnectionLifetime Property

Used to specify the maximum time during which an opened connection can be used by connection pool.

Class
TPoolingOptions

Syntax

```
property ConnectionLifetime: integer default DefValConnectionLifetime;
```

Remarks

Use the ConnectionLifeTime property to specify the maximum time during which an opened connection can be used by connection pool. Measured in milliseconds. Pool deletes connections with exceeded connection lifetime when TCustomDACConnection is about to close. If the ConnectionLifetime property is set to 0 (by default), then the lifetime of connection is infinity. ConnectionLifetime concerns only inactive connections in the pool.
5.11.1.22.2.3 MaxPoolSize Property

Used to specify the maximum number of connections that can be opened in the connection pool.

Class
TPoolingOptions

Syntax

\texttt{property MaxPoolSize: integer default DefValMaxPoolSize;}

Remarks
Specifies the maximum number of connections that can be opened in connection pool. Once this value is reached, no more connections are opened. The valid values are 1 and higher.

5.11.1.22.2.4 MinPoolSize Property

Used to specify the minimum number of connections that can be opened in the connection pool.

Class
TPoolingOptions

Syntax

\texttt{property MinPoolSize: integer default DefValMinPoolSize;}

Remarks
Use the MinPoolSize property to specify the minimum number of connections that can be opened in the connection pool.

5.11.1.22.2.4 Validate Property

Used for a connection to be validated when it is returned from the pool.

Class
TPoolingOptions

Syntax

\texttt{property Validate: boolean default DefValValidate;}

© 1997-2021 Devart. All Rights Reserved.
Remarks
If the Validate property is set to True, connection will be validated when it is returned from the pool. By default this option is set to False and pool does not validate connection when it is returned to be used by a TCustomDACConnection component.

5.11.1.23 TSmartFetchOptions Class

Smart fetch options are used to set up the behavior of the SmartFetch mode.

For a list of all members of this type, see TSmartFetchOptions members.

Unit
DBAccess

Syntax

TSmartFetchOptions = class(TPersistent);

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Sets SmartFetch mode enabled or not.</td>
</tr>
<tr>
<td>LiveBlock</td>
<td>Used to minimize memory consumption.</td>
</tr>
<tr>
<td>PrefetchedFields</td>
<td>List of fields additional to key fields that will be read from the database on dataset open.</td>
</tr>
<tr>
<td>SQLGetKeyValues</td>
<td>SQL query for the read key and prefetched fields from the database.</td>
</tr>
</tbody>
</table>
5.11.1.23.2 Properties

Properties of the **TSmartFetchOptions** class.

For a complete list of the **TSmartFetchOptions** class members, see the **TSmartFetchOptions Members** topic.

**Published**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enabled</strong></td>
<td>Sets SmartFetch mode enabled or not.</td>
</tr>
<tr>
<td><strong>LiveBlock</strong></td>
<td>Used to minimize memory consumption.</td>
</tr>
<tr>
<td><strong>PrefetchedFields</strong></td>
<td>List of fields additional to key fields that will be read from the database on dataset open.</td>
</tr>
<tr>
<td><strong>SQLGetKeyValues</strong></td>
<td>SQL query for the read key and prefetched fields from the database.</td>
</tr>
</tbody>
</table>

**See Also**

- **TSmartFetchOptions Class**
- **TSmartFetchOptions Class Members**

**5.11.1.23.2.1 Enabled Property**

Sets SmartFetch mode enabled or not.

**Class**

**TSmartFetchOptions**

**Syntax**

```plaintext
property Enabled: Boolean default False;
```
5.11.1.23.2.2 LiveBlock Property

Used to minimize memory consumption.

Class

**TSmartFetchOptions**

Syntax

```delphi
property LiveBlock: Boolean default True;
```

Remarks

If LiveBlock is True, then on navigating through a dataset forward or backward, memory will be allocated for records count defined in the the FetchRows propety, and no additional memory will be allocated. But if you return records that were read from the database before, they will be read from the database again, because when you left block with these records, memory was free. So the LiveBlock mode minimizes memory consumption, but can decrease performance, because it can lead to repeated data reading from the database.

The default value of LiveBlock is False.

5.11.1.23.2.3 PrefetchedFields Property

List of fields additional to key fields that will be read from the database on dataset open.

Class

**TSmartFetchOptions**

Syntax

```delphi
property PrefetchedFields: string;
```

Remarks

If you are going to use locate, filter or sort by some fields, then these fields should be added to the prefetched fields list to avoid excessive reading from the database.
5.11.1.23.2.4 SQLGetKeyValues Property

SQL query for the read key and prefetched fields from the database.

Class

**TSmartFetchOptions**

Syntax

```delphi
property SQLGetKeyValues: TStrings;
```

Remarks

SQLGetKeyValues is used when the basic SQL query is complex and the query for reading the key and prefetched fields can't be generated automatically.

5.11.2 Types

Types in the **DBAccess** unit.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAfterExecuteEvent</td>
<td>This type is used for the <code>TCustomDADataset.AfterExecute</code> and <code>TCustomDASQL.AfterExecute</code> events.</td>
</tr>
<tr>
<td>TAfterFetchEvent</td>
<td>This type is used for the <code>TCustomDADataset.AfterFetch</code> event.</td>
</tr>
<tr>
<td>TBeforeFetchEvent</td>
<td>This type is used for the <code>TCustomDADataset.BeforeFetch</code> event.</td>
</tr>
<tr>
<td>TConnectionLostEvent</td>
<td>This type is used for the <code>TCustomDADConnection.OnConnectionLost</code> event.</td>
</tr>
</tbody>
</table>
**TDACSErrorEvent**  
This type is used for the **TCustomDADCConnection.OnError** event.

**TDATransactionErrorEvent**  
This type is used for the **TDATransaction.OnError** event.

**TRefreshOptions**  
Represents the set of **TRefreshOption**.

**TUpdateExecuteEvent**  
This type is used for the **TCustomDADataSet.AfterUpdateExecute** and **TCustomDADataSet.BeforeUpdateExecute** events.

### 5.11.2.1 TAfterExecuteEvent Procedure Reference

This type is used for the **TCustomDADataSet.AfterExecute** and **TCustomDASQL.AfterExecute** events.

**Unit**

**DBAccess**

**Syntax**

```delphi
TAfterExecuteEvent = procedure (Sender: TObject; Result: boolean) of object;
```

**Parameters**

**Sender**

An object that raised the event.

**Result**

The result is True if SQL statement is executed successfully. False otherwise.
5.11.2.2 TAfterFetchEvent Procedure Reference

This type is used for the TCustomDADataSet.AfterFetch event.

Unit
DBAccess

Syntax

TAfterFetchEvent = procedure (DataSet: TCustomDADataSet) of object;

Parameters

DataSet
Holds the TCustomDADataSet descendant to synchronize the record position with.

5.11.2.3 TBeforeFetchEvent Procedure Reference

This type is used for the TCustomDADataSet.BeforeFetch event.

Unit
DBAccess

Syntax

TBeforeFetchEvent = procedure (DataSet: TCustomDADataSet; var Cancel: boolean) of object;

Parameters

DataSet
Holds the TCustomDADataSet descendant to synchronize the record position with.

Cancel
True, if the current fetch operation should be aborted.
5.11.2.4 TConnectionLostEvent Procedure Reference

This type is used for the TCustomDAConnection.OnConnectionLost event.

Unit
DBAccess

Syntax

\[
\text{TConnectionLostEvent} = \text{procedure} \ (\text{Sender}: \text{TObject}; \ \text{Component}: \text{TComponent}; \ \text{ConnLostCause}: \ \text{TConnLostCause}; \ \text{var} \ \text{RetryMode}: \ \text{TRetryMode}) \ \text{of} \ \text{object};
\]

Parameters

- **Sender**: An object that raised the event.
- **Component**: The object that the connection was made to.
- **ConnLostCause**: The reason of the connection loss.
- **RetryMode**: The application behavior when connection is lost.

5.11.2.5 TDACConnectionErrorEvent Procedure Reference

This type is used for the TCustomDAConnection.OnError event.

Unit
DBAccess

Syntax

\[
\text{TDACConnectionErrorEvent} = \text{procedure} \ (\text{Sender}: \text{TObject}; \ \text{E}: \ \text{EDAError}; \ \text{var} \ \text{Fail}: \ \text{boolean}) \ \text{of} \ \text{object};
\]

Parameters

- **Sender**: An object that raised the event.
- **E**: The error information.
- **Fail**: Indicates whether the error was handled successfully.
False, if an error dialog should be prevented from being displayed and EAbort exception should be raised to cancel current operation.

5.11.2.6 TDATransactionErrorEvent Procedure Reference

This type is used for the TDATransaction.OnError event.

Unit
DBAccess

Syntax

TDATransactionErrorEvent = procedure (Sender: TObject; E: EDAError; var Fail: boolean) of object;

Parameters

Sender
An object that raised the event.

E
The error code.

Fail
False, if an error dialog should be prevented from being displayed and EAbort exception to cancel the current operation should be raised.

5.11.2.7 TRefreshOptions Set

Represents the set of TRefreshOption.

Unit
DBAccess

Syntax

TRefreshOptions = set of TRefreshOption;

© 1997-2021
Devart. All Rights Reserved.
5.11.2.8 **TUpdateExecuteEvent Procedure Reference**

This type is used for the TCustomDADataset.AfterUpdateExecute and TCustomDADataset.BeforeUpdateExecute events.

**Unit**

*DBAccess*

**Syntax**

```
TUpdateExecuteEvent = procedure (Sender: TDataSet; StatementTypes: TStatementTypes; Params: TDAParams) of object;
```

**Parameters**

- **Sender**
  - An object that raised the event.

- **StatementTypes**
  - Holds the type of the SQL statement being executed.

- **Params**
  - Holds the parameters with which the SQL statement will be executed.

© 1997-2021 Devart. All Rights Reserved.

## 5.11.3 Enumerations

Enumerations in the *DBAccess* unit.

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLabelSet</td>
<td>Sets the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>TLockMode</td>
<td>Specifies the lock mode.</td>
</tr>
<tr>
<td>TRefreshOption</td>
<td>Indicates when the editing record will be refreshed.</td>
</tr>
<tr>
<td>TRetryMode</td>
<td>Specifies the application behavior when connection is lost.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
5.11.3.1 TLabelSet Enumeration

Sets the language of labels in the connect dialog.

Unit

DBAccess

Syntax

```
TLabelSet = (lsCustom, lsEnglish, lsFrench, lsGerman, lsItalian, lsPolish, lsPortuguese, lsRussian, lsSpanish);
```

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsCustom</td>
<td>Set the language of labels in the connect dialog manually.</td>
</tr>
<tr>
<td>lsEnglish</td>
<td>Set English as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsFrench</td>
<td>Set French as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsGerman</td>
<td>Set German as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsItalian</td>
<td>Set Italian as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsPolish</td>
<td>Set Polish as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsPortuguese</td>
<td>Set Portuguese as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsRussian</td>
<td>Set Russian as the language of labels in the connect dialog.</td>
</tr>
<tr>
<td>lsSpanish</td>
<td>Set Spanish as the language of labels in the connect dialog.</td>
</tr>
</tbody>
</table>

5.11.3.2 TLockMode Enumeration

Specifies the lock mode.

Unit

DBAccess

Syntax

```
TLockMode = (lmNone, lmPessimistic, lmOptimistic);
```

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lmNone</td>
<td></td>
</tr>
<tr>
<td>lmPessimistic</td>
<td></td>
</tr>
<tr>
<td>lmOptimistic</td>
<td></td>
</tr>
</tbody>
</table>
### TRefreshOption Enumeration

Indicates when the editing record will be refreshed.

**Unit**

DBAccess

**Syntax**

```plaintext
TRefreshOption = (roAfterInsert, roAfterUpdate, roBeforeEdit);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>roAfterInsert</td>
<td>Refresh is performed after inserting.</td>
</tr>
<tr>
<td>roAfterUpdate</td>
<td>Refresh is performed after updating.</td>
</tr>
<tr>
<td>roBeforeEdit</td>
<td>Refresh is performed by Edit method.</td>
</tr>
</tbody>
</table>

### TRetryMode Enumeration

Specifies the application behavior when connection is lost.

**Unit**

DBAccess

**Syntax**

```plaintext
TRetryMode = (rmRaise, rmReconnect, rmReconnectExecute);
```
### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmRaise</td>
<td>An exception is raised.</td>
</tr>
<tr>
<td>rmReconnect</td>
<td>Reconnect is performed and then exception is raised.</td>
</tr>
<tr>
<td>rmReconnectExec</td>
<td>Reconnect is performed and abortive operation is reexecuted. Exception is not raised.</td>
</tr>
</tbody>
</table>

### 5.11.4 Variables

Variables in the `DBAccess` unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQLOldBehavior</td>
<td>After assigning SQL text and modifying it by <code>AddWhere</code>, <code>DeleteWhere</code>, and <code>SetOrderBy</code>, all subsequent changes of the SQL property will not be reflected in the <code>BaseSQL</code> property.</td>
</tr>
<tr>
<td>ChangeCursor</td>
<td>When set to True allows data access components to change screen cursor for the execution time.</td>
</tr>
<tr>
<td>SQLGeneratorCompatibility</td>
<td>The value of the <code>TCustomDADataSet.BaseSQL</code> property is used to complete the refresh SQL statement, if the manually assigned <code>TCustomDAUpdateSQL.RefreshSQL</code> property contains only WHERE clause.</td>
</tr>
</tbody>
</table>
5.11.4.1 BaseSQLOldBehavior Variable

After assigning SQL text and modifying it by AddWhere, DeleteWhere, and SetOrderBy, all subsequent changes of the SQL property will not be reflected in the BaseSQL property.

Unit

DBAccess

Syntax

BaseSQLOldBehavior: boolean = False;

Remarks

The BaseSQL property is similar to the SQL property, but it does not store changes made by the AddWhere, DeleteWhere, and SetOrderBy methods. After assigning SQL text and modifying it by one of these methods, all subsequent changes of the SQL property will not be reflected in the BaseSQL property. This behavior was changed in PgDAC. To restore old behavior, set the BaseSQLOldBehavior variable to True.

5.11.4.2 ChangeCursor Variable

When set to True allows data access components to change screen cursor for the execution time.

Unit

DBAccess

Syntax

ChangeCursor: boolean = True;

5.11.4.3 SQLGeneratorCompatibility Variable

The value of the TCustomDADataSet.BaseSQL property is used to complete the refresh SQL statement, if the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause.
Unit

DBAccess

Syntax

SQLGeneratorCompatibility: boolean = False;

Remarks

If the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause, PgDAC uses the value of the TCustomDADataset.BaseSQL property to complete the refresh SQL statement. In this situation all modifications applied to the SELECT query by functions TCustomDADataset.AddWhere, TCustomDADataset.DeleteWhere are not taken into account. This behavior was changed in PgDAC. To restore the old behavior, set the BaseSQLOldBehavior variable to True.

© 1997-2021 Devart. All Rights Reserved.

5.12 MemData

This unit contains classes for storing data in memory.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAttribute</td>
<td>TAttribute is not used in PgDAC.</td>
</tr>
<tr>
<td>TBlob</td>
<td>Holds large object value for field and parameter dtBlob, dtMemo data types.</td>
</tr>
<tr>
<td>TCompressedBlob</td>
<td>Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.</td>
</tr>
<tr>
<td>TDBObject</td>
<td>A base class for classes that work with user-defined data types that have attributes.</td>
</tr>
<tr>
<td>TMemData</td>
<td>Implements in-memory database.</td>
</tr>
<tr>
<td>TObjectType</td>
<td>This class is not used.</td>
</tr>
</tbody>
</table>
### TSharedObject

A base class that allows to simplify memory management for object referenced by several other objects.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLocateExOptions</td>
<td>Represents the set of TLocateExOption.</td>
</tr>
<tr>
<td>TUpdateRecKinds</td>
<td>Represents the set of TUpdateRecKind.</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCompressBlobMode</td>
<td>Specifies when the values should be compressed and the way they should be stored.</td>
</tr>
<tr>
<td>TConnLostCause</td>
<td>Specifies the cause of the connection loss.</td>
</tr>
<tr>
<td>TDANumericType</td>
<td>Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.</td>
</tr>
<tr>
<td>TLocateExOption</td>
<td>Allows to set additional search parameters which will be used by the LocateEx method.</td>
</tr>
<tr>
<td>TSortType</td>
<td>Specifies a sort type for string fields.</td>
</tr>
<tr>
<td>TUpdateRecKind</td>
<td>Indicates records for which the ApplyUpdates method will be performed.</td>
</tr>
</tbody>
</table>
5.12.1 Classes

Classes in the **MemData** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAttribute</td>
<td>TAttribute is not used in PgDAC.</td>
</tr>
<tr>
<td>TBlob</td>
<td>Holds large object value for field and parameter dtBlob, dtMemo data types.</td>
</tr>
<tr>
<td>TCompressedBlob</td>
<td>Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.</td>
</tr>
<tr>
<td>TDBObject</td>
<td>A base class for classes that work with user-defined data types that have attributes.</td>
</tr>
<tr>
<td>TMemData</td>
<td>Implements in-memory database.</td>
</tr>
<tr>
<td>TObjectType</td>
<td>This class is not used.</td>
</tr>
<tr>
<td>TSharedObject</td>
<td>A base class that allows to simplify memory management for object referenced by several other objects.</td>
</tr>
</tbody>
</table>

5.12.1.1 TAttribute Class

TAttribute is not used in PgDAC.

For a list of all members of this type, see TAttribute members.

Unit

**MemData**

Syntax

```cpp
TAttribute = class (System.TObject);
```
### 5.12.1.1.1 Members

**TAttribute** class overview.

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeNo</td>
<td>Returns an attribute’s ordinal position in object.</td>
</tr>
<tr>
<td>DataSize</td>
<td>Returns the size of an attribute value in internal representation.</td>
</tr>
<tr>
<td>DataType</td>
<td>Returns the type of data that was assigned to the Attribute.</td>
</tr>
<tr>
<td>Length</td>
<td>Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Returns a TObjectType object for an object attribute.</td>
</tr>
<tr>
<td>Offset</td>
<td>Returns an offset of the attribute value in internal representation.</td>
</tr>
<tr>
<td>Owner</td>
<td>Indicates TObjectType that uses the attribute to represent one of its attributes.</td>
</tr>
<tr>
<td>Scale</td>
<td>Returns the scale of dtFloat and dtInteger attributes.</td>
</tr>
<tr>
<td>Size</td>
<td>Returns the size of an attribute value in external representation.</td>
</tr>
</tbody>
</table>
5.12.1.2 Properties

Properties of the TAttribute class.

For a complete list of the TAttribute class members, see the TAttribute Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeNo</td>
<td>Returns an attribute's ordinal position in object.</td>
</tr>
<tr>
<td>DataSize</td>
<td>Returns the size of an attribute value in internal representation.</td>
</tr>
<tr>
<td>DataType</td>
<td>Returns the type of data that was assigned to the Attribute.</td>
</tr>
<tr>
<td>Length</td>
<td>Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Returns a TObjectType object for an object attribute.</td>
</tr>
<tr>
<td>Offset</td>
<td>Returns an offset of the attribute value in internal representation.</td>
</tr>
<tr>
<td>Owner</td>
<td>Indicates TObjectType that uses the attribute to represent one of its attributes.</td>
</tr>
<tr>
<td>Scale</td>
<td>Returns the scale of dtFloat and dtInteger attributes.</td>
</tr>
<tr>
<td>Size</td>
<td>Returns the size of an attribute value in external representation.</td>
</tr>
</tbody>
</table>

See Also
- TAttribute Class
- TAttribute Class Members
5.12.1.2.1 AttributeNo Property

Returns an attribute's ordinal position in object.

Class
TAttribute

Syntax

```
property AttributeNo: Word;
```

Remarks
Use the AttributeNo property to learn an attribute's ordinal position in object, where 1 is the first field.

See Also
- TObjectType.Attributes

5.12.1.2.2 DataSize Property

Returns the size of an attribute value in internal representation.

Class
TAttribute

Syntax

```
property DataSize: Integer;
```

Remarks
Use the DataSize property to learn the size of an attribute value in internal representation.
5.12.1.1.2.3  **DataType Property**

Returns the type of data that was assigned to the Attribute.

**Class**

`TAttribute`

**Syntax**

```delphi
property DataType: Word;
```

**Remarks**

Use the `DataType` property to discover the type of data that was assigned to the Attribute.

Possible values: `dtDate`, `dtFloat`, `dtInteger`, `dtString`, `dtObject`.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.1.2.4  **Length Property**

Returns the length of the string for `dtString` attribute and precision for `dtInteger` and `dtFloat` attribute.

**Class**

`TAttribute`

**Syntax**

```delphi
property Length: Word;
```

**Remarks**

Use the `Length` property to learn the length of the string for `dtString` attribute and precision for `dtInteger` and `dtFloat` attribute.

**See Also**

- `Scale`
5.12.1.1.2.5 ObjectType Property

Returns a TObjectType object for an object attribute.

Class

TAttribute

Syntax

```delphi
property ObjectType: TObjectType;
```

Remarks

Use the ObjectType property to return a TObjectType object for an object attribute.

© 1997-2021
Devart. All Rights Reserved.

5.12.1.1.2.6 Offset Property

Returns an offset of the attribute value in internal representation.

Class

TAttribute

Syntax

```delphi
property Offset: Integer;
```

Remarks

Use the DataSize property to learn an offset of the attribute value in internal representation.

© 1997-2021
Devart. All Rights Reserved.

5.12.1.1.2.7 Owner Property

Indicates TObjectType that uses the attribute to represent one of its attributes.

Class

TAttribute
5.12.1.1.2.8  Scale Property

Returns the scale of dtFloat and dtInteger attributes.

Class
TAttribute

Syntax

```pascal
property Scale: Word;
```

Remarks
Use the Scale property to learn the scale of dtFloat and dtInteger attributes.

See Also
• Length

5.12.1.1.2.9  Size Property

Returns the size of an attribute value in external representation.

Class
TAttribute

Syntax

```pascal
property Size: Integer;
```
Remarks
Read Size to learn the size of an attribute value in external representation.

For example:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>dtDate</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>(sizeof(TDateTime))</td>
</tr>
<tr>
<td>dtFloat</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>(sizeof(Double))</td>
</tr>
<tr>
<td>dtInteger</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>(sizeof(Integer))</td>
</tr>
</tbody>
</table>

See Also
- DataSize

Unit
MemData

Syntax

```pascal
TBlob = class(TSharedObject);
```

Remarks
Object TBlob holds large object value for the field and parameter dtBlob, dtMemo, dtWideMemo data types.

Inheritance Hierarchy

```
TSharedObject
  TBlob
```

See Also
5.12.1.2.1  Members

**TBlob** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td>AsWideString</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>IsUnicode</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td>RefCount (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size</td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from <strong>TSharedObject</strong>)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td>Clear</td>
<td>Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Loads the contents of a file into a TBlob object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Copies the contents of a stream into the TBlob object.</td>
</tr>
<tr>
<td>Read</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlob.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
<tr>
<td>SaveToFile</td>
<td>Saves the contents of the TBlob object to a file.</td>
</tr>
<tr>
<td>SaveToStream</td>
<td>Copies the contents of a TBlob object to a stream.</td>
</tr>
<tr>
<td>Truncate</td>
<td>Sets new TBlob size and discards all data over it.</td>
</tr>
<tr>
<td>Write</td>
<td>Stores a raw sequence of bytes into a TBlob object.</td>
</tr>
</tbody>
</table>

Properties of the TBlob class.

For a complete list of the TBlob class members, see the [TBlob Members](#) topic.

## Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td>AsWideString</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>IsUnicode</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size</td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
</tbody>
</table>

See Also

- TBlob Class
- TBlob Class Members
### 5.12.1.2.2.1 AsString Property

Used to manipulate BLOB value as string.

**Class**

**TBlob**

**Syntax**

```delphi
property AsString: string;
```

**Remarks**

Use the AsString property to manipulate BLOB value as string.

**See Also**

- **Assign**
- **AsWideString**

### 5.12.1.2.2.2 AsWideString Property

Used to manipulate BLOB value as Unicode string.

**Class**

**TBlob**

**Syntax**

```delphi
property AsWideString: string;
```

**Remarks**

Use the AsWideString property to manipulate BLOB value as Unicode string.

**See Also**

- **Assign**
- **AsString**
5.12.1.2.2.3 IsUnicode Property

Gives choice of making TBlob store and process data in Unicode format or not.

Class
TBlob

Syntax

property IsUnicode: boolean;

Remarks
Set IsUnicode to True if you want TBlob to store and process data in Unicode format.

Note: changing this property raises an exception if TBlob is not empty.

5.12.1.2.2.4 Size Property

Used to learn the size of the TBlob value in bytes.

Class
TBlob

Syntax

property Size: Cardinal;

Remarks
Use the Size property to find out the size of the TBlob value in bytes.
Methods of the **TBlob** class.

For a complete list of the **TBlob** class members, see the [TBlob Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from <strong>TSharedObject</strong>)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td>Clear</td>
<td>Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Loads the contents of a file into a TBlob object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Copies the contents of a stream into the TBlob object.</td>
</tr>
<tr>
<td>Read</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlob.</td>
</tr>
<tr>
<td>Release (inherited from <strong>TSharedObject</strong>)</td>
<td>Decrements the reference count.</td>
</tr>
<tr>
<td>SaveToFile</td>
<td>Saves the contents of the TBlob object to a file.</td>
</tr>
<tr>
<td>SaveToStream</td>
<td>Copies the contents of a TBlob object to a stream.</td>
</tr>
<tr>
<td>Truncate</td>
<td>Sets new TBlob size and discards all data over it.</td>
</tr>
<tr>
<td>Write</td>
<td>Stores a raw sequence of bytes into a TBlob object.</td>
</tr>
</tbody>
</table>

See Also
- [TBlob Class](#)
- [TBlob Class Members](#)
5.12.1.2.3.1 Assign Method

Sets BLOB value from another TBlob object.

Class

TBlob

Syntax

```
procedure Assign(Source: TBlob);
```

Parameters

Source

Holds the BLOB from which the value to the current object will be assigned.

Remarks

Call the Assign method to set BLOB value from another TBlob object.

See Also

- LoadFromStream
- AsString
- AsWideString

5.12.1.2.3.2 Clear Method

Deletes the current value in TBlob object.

Class

TBlob

Syntax

```
procedure Clear; virtual;
```

Remarks

Call the Clear method to delete the current value in TBlob object.
5.12.1.2.3.3  LoadFromFile Method

Loads the contents of a file into a TBlob object.

Class

TBlob

Syntax

procedure LoadFromFile(const FileName: string);

Parameters

FileName

Holds the name of the file from which the TBlob value is loaded.

Remarks

Call the LoadFromFile method to load the contents of a file into a TBlob object. Specify the
name of the file to load into the field as the value of the FileName parameter.

See Also

• SaveToFile

5.12.1.2.3.4  LoadFromStream Method

Copies the contents of a stream into the TBlob object.

Class

TBlob

Syntax

procedure LoadFromStream(Stream: TStream); virtual;

Parameters

Stream

Holds the specified stream from which the field's value is copied.

Remarks
Call the LoadFromStream method to copy the contents of a stream into the TBlob object. Specify the stream from which the field's value is copied as the value of the Stream parameter.

See Also

- SaveToStream

Class

TBlob

Syntax

```pascal
function Read(Position: Cardinal; Count: Cardinal; Dest: IntPtr): Cardinal; virtual;
```

Parameters

- **Position**
  
  Holds the starting point of the byte sequence.

- **Count**
  
  Holds the size of the sequence in bytes.

- **Dest**
  
  Holds a pointer to the memory area where to store the sequence.

Return Value

- Actually read byte count if the sequence crosses object size limit.

Remarks

Call the Read method to acquire a raw sequence of bytes from the data stored in TBlob.

The Position parameter is the starting point of byte sequence which lasts Count number of bytes. The Dest parameter is a pointer to the memory area where to store the sequence.

If the sequence crosses object size limit, function will return actually read byte count.

See Also

- Write
5.12.1.2.3.6  SaveToFile Method

Saves the contents of the TBlob object to a file.

Class

TBlob

Syntax

procedure SaveToFile(const FileName: string);

Parameters

FileName
Holds a string that contains the name of the file.

Remarks

Call the SaveToFile method to save the contents of the TBlob object to a file. Specify the name of the file as the value of the FileName parameter.

See Also

• LoadFromFile

5.12.1.2.3.7  SaveToStream Method

Copies the contents of a TBlob object to a stream.

Class

TBlob

Syntax

procedure SaveToStream(Stream: TStream); virtual;

Parameters

Stream
Holds the name of the stream.
Remarks

Call the SaveToStream method to copy the contents of a TBlob object to a stream. Specify the name of the stream to which the field's value is saved as the value of the Stream parameter.

See Also

• LoadFromStream

5.12.1.2.3.8  Truncate Method

Sets new TBlob size and discards all data over it.

Class

TBlob

Syntax

procedure Truncate(NewSize: Cardinal); virtual;

Parameters

NewSize

Holds the new size of TBlob.

Remarks

Call the Truncate method to set new TBlob size and discard all data over it. If NewSize is greater or equal TBlob.Size, it does nothing.

5.12.1.2.3.9  Write Method

Stores a raw sequence of bytes into a TBlob object.

Class

TBlob
Syntax

```pascal
procedure Write(Position: Cardinal; Count: Cardinal; Source: IntPtr); virtual;
```

**Parameters**

- **Position**
  - Holds the starting point of the byte sequence.

- **Count**
  - Holds the size of the sequence in bytes.

- **Source**
  - Holds a pointer to a source memory area.

**Remarks**

Call the Write method to store a raw sequence of bytes into a TBlob object.

The Position parameter is the starting point of byte sequence which lasts Count number of bytes. The Source parameter is a pointer to a source memory area.

If the value of the Position parameter crosses current size limit of TBlob object, source data will be appended to the object data.

**See Also**

- [Read](#)

© 1997-2021 Devart. All Rights Reserved.

5.12.1.3 **TCompressedBlob Class**

Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.

For a list of all members of this type, see [TCompressedBlob](#) members.

**Unit**

MemData

**Syntax**

```pascal
TCompressedBlob = class(TBlob);
```
Remarks

TCompressedBlob is a descendant of the TBlob class. It holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data. For more information about using BLOB compression see TCustomDADataSet.Options.

Note: Internal compression functions are available in CodeGear Delphi 2007 for Win32, Borland Developer Studio 2006, Borland Delphi 2005, and Borland Delphi 7. To use BLOB compression under Borland Delphi 6 and Borland C++ Builder you should use your own compression functions. To use them set the CompressProc and UncompressProc variables declared in the MemUtils unit.

Example

```pascal
type
  TCompressProc = function(dest: IntPtr; destLen: IntPtr; const source: IntPtr; sourceLen: longint): longint;
  TUncompressProc = function(dest: IntPtr; destlen: IntPtr; source: IntPtr; sourceLen: longint): longint;

var
  CompressProc: TCompressProc;
  UncompressProc: TUncompressProc;
```

Inheritance Hierarchy

TSharedObject
  TBlob
    TCompressedBlob

See Also

- TBlob
- TMemDataSet.GetBlob
- TCustomDADataSet.Options

© 1997-2021
Devart. All Rights Reserved.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsWideString</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>Compressed</td>
<td>Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td>CompressedSize</td>
<td>Used to indicate compressed size of the Blob data.</td>
</tr>
<tr>
<td>IsUnicode</td>
<td>Gives choice of making TBlog store and process data in Unicode format or not.</td>
</tr>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size</td>
<td>Used to learn the size of the TBlog value in bytes.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Sets BLOB value from another TBlog object.</td>
</tr>
<tr>
<td>Clear</td>
<td>Deletes the current value in TBlog object.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>Loads the contents of a file into a TBlog object.</td>
</tr>
<tr>
<td>LoadFromStream</td>
<td>Copies the contents of a stream into the TBlog object.</td>
</tr>
<tr>
<td>Read</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlog.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
<tr>
<td>SaveToFile</td>
<td>Saves the contents of the TBlog object to a file.</td>
</tr>
<tr>
<td>SaveToStream</td>
<td>Copies the contents of a TBlog object to a stream.</td>
</tr>
<tr>
<td>Truncate</td>
<td>Sets new TBlog size and</td>
</tr>
</tbody>
</table>
## 5.12.1.3.2 Properties

Properties of the `TCompressedBlob` class.

For a complete list of the `TCompressedBlob` class members, see the [TCompressedBlob Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AsString</code> (inherited from <code>TBlob</code>)</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td><code>AsWideString</code> (inherited from <code>TBlob</code>)</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td><code>Compressed</code></td>
<td>Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td><code>CompressedSize</code></td>
<td>Used to indicate compressed size of the Blob data.</td>
</tr>
<tr>
<td><code>IsUnicode</code> (inherited from <code>TBlob</code>)</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td><code>RefCount</code> (inherited from <code>TSharedObject</code>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td><code>Size</code> (inherited from <code>TBlob</code>)</td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
</tbody>
</table>

### See Also

- `TCompressedBlob Class`
- `TCompressedBlob Class Members`
5.12.1.3.2.1  Compressed Property

Used to indicate if the Blob is compressed.

Class

TCompressedBlob

Syntax

`property` Compressed: boolean;

Remarks

Indicates whether the Blob is compressed. Set this property to True or False to compress or decompress the Blob.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.3.2.2  CompressedSize Property

Used to indicate compressed size of the Blob data.

Class

TCompressedBlob

Syntax

`property` CompressedSize: Cardinal;

Remarks

Indicates compressed size of the Blob data.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.4  TDBObject Class

A base class for classes that work with user-defined data types that have attributes.

For a list of all members of this type, see TDBObject members.

Unit
MemData

Syntax

```pascal
TDBObject = class(TSharedObject);
```

Remarks

TDBObject is a base class for classes that work with user-defined data types that have attributes.

Inheritance Hierarchy

- **TSharedObject**
- **TDBObject**

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefCount</td>
<td>(inherited from TSharedObject) Used to return the count of reference to a</td>
</tr>
<tr>
<td></td>
<td>TSharedObject object.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>(inherited from TSharedObject) Increments the reference count for the number</td>
</tr>
<tr>
<td></td>
<td>of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Release</td>
<td>(inherited from TSharedObject) Decrements the reference count.</td>
</tr>
</tbody>
</table>
5.12.1.5  TMemData Class

Implements in-memory database.

For a list of all members of this type, see TMemData members.

Unit

MemData

Syntax

TMemData = class(TData);

Inheritance Hierarchy

TData
    TMemData

5.12.1.6  TObjectType Class

This class is not used.

For a list of all members of this type, see TObjectType members.

Unit

MemData

Syntax

TObjectType = class(TSharedObject);

Inheritance Hierarchy

TSharedObject
5.12.1.6.1 Members

**TObjectType** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeCount</td>
<td>Used to indicate the number of attributes of type.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Used to access separate attributes.</td>
</tr>
<tr>
<td>DataType</td>
<td>Used to indicate the type of object dtObject, dtArray or dtTable.</td>
</tr>
<tr>
<td>RefCount</td>
<td>(inherited from <strong>TSharedObject</strong>) Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size</td>
<td>Used to learn the size of an object instance.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>(inherited from <strong>TSharedObject</strong>) Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>FindAttribute</td>
<td>Indicates whether a specified Attribute component is referenced in the TAttributes object.</td>
</tr>
<tr>
<td>Release</td>
<td>(inherited from <strong>TSharedObject</strong>) Decrements the reference count.</td>
</tr>
</tbody>
</table>
5.12.1.6.2 Properties

Properties of the TObjectType class.

For a complete list of the TObjectType class members, see the TObjectType Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeCount</td>
<td>Used to indicate the number of attributes of type.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Used to access separate attributes.</td>
</tr>
<tr>
<td>DataType</td>
<td>Used to indicate the type of object dtObject, dtArray or dtTable.</td>
</tr>
<tr>
<td>RefCount</td>
<td>(inherited from TSharedObject) Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size</td>
<td>Used to learn the size of an object instance.</td>
</tr>
</tbody>
</table>

See Also

- TObjectType Class
- TObjectType Class Members

© 1997-2021 Devart. All Rights Reserved.

5.12.1.6.2.1 AttributeCount Property

Used to indicate the number of attributes of type.

Class

TObjectClass

Syntax

```
property AttributeCount: Integer;
```

Remarks
Use the AttributeCount property to determine the number of attributes of type.

5.12.1.6.2.2  Attributes Property(Indexer)

Used to access separate attributes.

Class

TObjectType

Syntax

property Attributes[Index: integer]: TAttribute;

Parameters

Index

Holds the attribute's ordinal position.

Remarks

Use the Attributes property to access individual attributes. The value of the Index parameter corresponds to the AttributeNo property of TAttribute.

See Also

• TAttribute
• FindAttribute

5.12.1.6.2.3  DataType Property

Used to indicate the type of object dtObject, dtArray or dtTable.

Class

TObjectType

Syntax

property DataType: Word;
Remarks

Use the DataType property to determine the type of object dtObject, dtArray or dtTable.

Class

TObjectType

Syntax

```pascal
property Size: Integer;
```

Remarks

Use the Size property to find out the size of an object instance. Size is a sum of all attribute sizes.

See Also

- `TAttribute.Size`

Methods

Methods of the TObjectType class.

For a complete list of the TObjectType class members, see the TObjectType Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
</tbody>
</table>
Indicates whether a specified Attribute component is referenced in the TAttributes object.

**Class**

*TObjectType*

**Syntax**

```plaintext
function FindAttribute(const Name: string): TAttribute; virtual;
```

**Parameters**

*Name*

Holds the name of the attribute to search for.

**Return Value**

TAttribute, if an attribute with a matching name was found. Nil Otherwise.

**Remarks**

Call FindAttribute to determine if a specified Attribute component is referenced in the TAttributes object. Name is the name of the Attribute for which to search. If FindAttribute finds an Attribute with a matching name, it returns the TAttribute. Otherwise it returns nil.

**See Also**

- *TAttribute*
- *Attributes*
5.12.1.7 TSharedObject Class

A base class that allows to simplify memory management for object referenced by several other objects.

For a list of all members of this type, see TSharedObject members.

Unit

MemData

Syntax

TSharedObject = class(System.TObject);

Remarks

TSharedObject allows to simplify memory management for object referenced by several other objects. TSharedObject holds a count of references to itself. When any object (referer object) is going to use TSharedObject, it calls the TSharedObject.AddRef method. Referer object has to call the TSharedObject.Release method after using TSharedObject.

See Also

- TBlob
- TObjectType

5.12.1.7.1 Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

Methods
5.12.1.7.2 Properties

Properties of the `TSharedObject` class.

For a complete list of the `TSharedObject` class members, see the `TSharedObject Members` topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

See Also
- `TSharedObject Class`
- `TSharedObject Class Members`

5.12.1.7.2.1 RefCount Property

Used to return the count of reference to a TSharedObject object.

Class

`TSharedObject`

Syntax

```pascal
property RefCount: Integer;
```
Remarks

Returns the count of reference to a TSharedObject object.

5.12.1.7.3 Methods

Methods of the TSharedObject class.

For a complete list of the TSharedObject class members, see the TSharedObject Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

See Also

- TSharedObject Class
- TSharedObject Class Members

5.12.1.7.3.1 AddRef Method

Increments the reference count for the number of references dependent on the TSharedObject object.

Class

TSharedObject

Syntax

```pascal
procedure AddRef;
```
Remarks
Increments the reference count for the number of references dependent on the TSharedObject object.

See Also
- Release

Class
TSharedObject

Syntax
procedure Release;

Remarks
Call the Release method to decrement the reference count. When RefCount is 1, TSharedObject is deleted from memory.

See Also
- AddRef

5.12.2 Types

Types in the MemData unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLocateExOptions</td>
<td>Represents the set of TLocateExOption.</td>
</tr>
</tbody>
</table>
5.12.2.1 TLocateExOptions Set

Represents the set of TLocateExOption.

Unit

MemData

Syntax

TLocateExOptions = set of TLocateExOption;

5.12.2.2 TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

Unit

MemData

Syntax

TUpdateRecKinds = set of TUpdateRecKind;

5.12.3 Enumerations

Enumerations in the MemData unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCompressBlobMode</td>
<td>Specifies when the values should be compressed and</td>
</tr>
</tbody>
</table>
5.12.3.1 TCompressBlobMode Enumeration

Specifies when the values should be compressed and the way they should be stored.

Unit
MemData

Syntax
TCompressBlobMode = (cbNone, cbClient, cbServer, cbClientServer);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbClient</td>
<td>Values are compressed and stored as compressed data at the client side. Before posting data to the server decompression is performed and data at the server side stored in the original form. Allows to reduce used client memory due to increase access time to field values. The time spent on the opening DataSet and executing Post increases.</td>
</tr>
<tr>
<td>cbClientServer</td>
<td>Values are compressed and stored in compressed form. Allows to decrease the volume of used memory at client and server</td>
</tr>
</tbody>
</table>
sides. Access time to the field values increases as for cbClient. The time spent on opening DataSet and executing Post decreases. **Note:** On using cbServer or cbClientServer data on the server is stored as compressed. Other applications can add records in uncompressed format but can’t read and write already compressed data. If compressed BLOB is partially changed by another application (if signature was not changed), DAC will consider its value as NULL. Blob compression is not applied to Memo fields because of possible cutting.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbNone</td>
<td>Values not compressed. The default value.</td>
</tr>
<tr>
<td>cbServer</td>
<td>Values are compressed before passing to the server and store at the server in compressed form. Allows to decrease database size on the server. Access time to the field values does not change. The time spent on opening DataSet and executing Post usually decreases.</td>
</tr>
</tbody>
</table>

---

**5.12.3.2 TConnLostCause Enumeration**

Specifies the cause of the connection loss.

**Unit**

**MemData**

**Syntax**

```
TConnLostCause = (clUnknown, clExecute, clOpen, clRefresh, clApply, clServiceQuery, clTransStart, clConnectionApply, clConnect);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>clApply</td>
<td>Connection loss detected during DataSet.ApplyUpdates (Reconnect/Reexecute possible).</td>
</tr>
<tr>
<td>clConnect</td>
<td>Connection loss detected during connection establishing (Reconnect possible).</td>
</tr>
<tr>
<td>clConnectionApply</td>
<td>Connection loss detected during Connection.ApplyUpdates (Reconnect/Reexecute possible).</td>
</tr>
<tr>
<td>clExecute</td>
<td>Connection loss detected during SQL execution (Reconnect with exception is possible).</td>
</tr>
<tr>
<td>clOpen</td>
<td>Connection loss detected during execution of a SELECT</td>
</tr>
</tbody>
</table>
5.12.3.3 TDANumericType Enumeration

Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.

**Unit**

**MemData**

**Syntax**

```plaintext
TDANumericType = (ntFloat, ntBCD, ntFmtBCD);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntBCD</td>
<td>Data is stored on the client side as currency and represented as TBCDField. This format allows storing data with precision up to 0.0001.</td>
</tr>
<tr>
<td>ntFloat</td>
<td>Data stored on the client side is in double format and represented as TFloatField. The default value.</td>
</tr>
<tr>
<td>ntFmtBCD</td>
<td>Data is represented as TFMTBCDField. TFMTBCDField gives greater precision and accuracy than TBCDField, but it is slower.</td>
</tr>
</tbody>
</table>

5.12.3.4 TLocateExOption Enumeration

Allows to set additional search parameters which will be used by the LocateEx method.

**Unit**
### TLocateExOption Syntax

\[
\text{TLocateExOption} = (\text{lxCaseInsensitive}, \text{lxPartialKey}, \text{lxNearest}, \text{lxNext}, \text{lxUp}, \text{lxPartialCompare});
\]

### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lxCaseInsensitive</td>
<td>Similar to lCaseInsensitive. Key fields and key values are matched without regard to the case.</td>
</tr>
<tr>
<td>lxNearest</td>
<td>LocateEx moves the cursor to a specific record in a dataset or to the first record in the dataset that is greater than the values specified in the KeyValues parameter. For this option to work correctly dataset should be sorted by the fields the search is performed in. If dataset is not sorted, the function may return a line that is not connected with the search condition.</td>
</tr>
<tr>
<td>lxNext</td>
<td>LocateEx searches from the current record.</td>
</tr>
<tr>
<td>lxPartialCompare</td>
<td>Similar to lxPartialKey, but the difference is that it can process value entries in any position. For example, 'HAM' would match both 'HAMM', 'HAMMER.', and also 'MR HAMMER'.</td>
</tr>
<tr>
<td>lxPartialKey</td>
<td>Similar to lPartialKey. Key values can include only a part of the matching key field value. For example, 'HAM' would match both 'HAMM' and 'HAMMER.', but not 'MR HAMMER'.</td>
</tr>
<tr>
<td>lxUp</td>
<td>LocateEx searches from the current record to the first record.</td>
</tr>
</tbody>
</table>

5.12.3.5 **TSortType Enumeration**

Specifies a sort type for string fields.

### Unit Syntax

\[
\text{TSortType} = (\text{stCaseSensitive}, \text{stCaseInsensitive}, \text{stBinary});
\]

### Values
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>stBinary</td>
<td>Sorting by character ordinal values (this comparison is also case sensitive).</td>
</tr>
<tr>
<td>stCaseInsensitive</td>
<td>Sorting without case sensitivity.</td>
</tr>
<tr>
<td>stCaseSensitive</td>
<td>Sorting with case sensitivity.</td>
</tr>
</tbody>
</table>

5.12.3.6 TUpdateRecKind Enumeration

Indicates records for which the ApplyUpdates method will be performed.

Unit

MemData

Syntax

```
TUpdateRecKind = (ukUpdate, ukInsert, ukDelete);
```

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ukDelete</td>
<td>ApplyUpdates will be performed for deleted records.</td>
</tr>
<tr>
<td>ukInsert</td>
<td>ApplyUpdates will be performed for inserted records.</td>
</tr>
<tr>
<td>ukUpdate</td>
<td>ApplyUpdates will be performed for updated records.</td>
</tr>
</tbody>
</table>

5.13 MemDS

This unit contains implementation of the TMemDataSet class.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMemDataSet</td>
<td>A base class for working with data and manipulating data in memory.</td>
</tr>
</tbody>
</table>
Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoNotRaiseExceptionOnUaFail</td>
<td>An exception will be raised if the value of the UpdateAction parameter is uaFail.</td>
</tr>
<tr>
<td>SendDataSetChangeEventAfterOpen</td>
<td>The DataSetChangeEvent is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.</td>
</tr>
</tbody>
</table>

5.13.1 Classes

Classes in the MemDS unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMemDataSet</td>
<td>A base class for working with data and manipulating data in memory.</td>
</tr>
</tbody>
</table>

5.13.1.1 TMemDataSet Class

A base class for working with data and manipulating data in memory.

For a list of all members of this type, see TMemDataSet members.

Unit

MemDS

Syntax

```pascal
TMemDataSet = class(TDataSet);
```
Remarks

TMemDataSet derives from the TDataSet database-engine independent set of properties, events, and methods for working with data and introduces additional techniques to store and manipulate data in memory.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>Prepared</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>
# Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong></td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>Locate</strong></td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong></td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong></td>
<td>Marks all records in the cache of updates as</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnUpdateError</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
</tbody>
</table>
OnUpdateRecord | Occurs when a single update component can not handle the updates.

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.13.1.1.2 Properties

Properties of the TMemDataSet class.

For a complete list of the TMemDataSet class members, see the TMemDataSet Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>Prepared</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
</tbody>
</table>
5.13.1.2.1  CachedUpdates Property

Used to enable or disable the use of cached updates for a dataset.

Class

TMemDataSet

Syntax

| property CachedUpdates: boolean default False; |

Remarks

Use the CachedUpdates property to enable or disable the use of cached updates for a dataset. Setting CachedUpdates to True enables updates to a dataset (such as posting changes, inserting new records, or deleting records) to be stored in an internal cache on the client side instead of being written directly to the dataset's underlying database tables. When changes are completed, an application writes all cached changes to the database in the context of a single transaction.

Cached updates are especially useful for client applications working with remote database servers. Enabling cached updates brings up the following benefits:

- Fewer transactions and shorter transaction times.
- Minimized network traffic.

The potential drawbacks of enabling cached updates are:

- Other applications can access and change the actual data on the server while users are editing local copies of data, resulting in an update conflict when cached updates are applied to the database.
- Other applications cannot access data changes made by an application until its cached updates are applied to the database.
The default value is False.

**Note:** When establishing master/detail relationship the CachedUpdates property of detail dataset works properly only when `TDADatasetOptions.LocalMasterDetail` is set to True.

**See Also**
- `UpdatesPending`
- `TMemDataSet.ApplyUpdates`
- `RestoreUpdates`
- `CommitUpdates`
- `CancelUpdates`
- `UpdateStatus`
- `TCustomDADataSet.Options`

5.13.1.2.2  **IndexFieldNames Property**

Used to get or set the list of fields on which the recordset is sorted.

**Class**

`TMemDataSet`

**Syntax**

```plaintext
property IndexFieldNames: string;
```

**Remarks**

Use the `IndexFieldNames` property to get or set the list of fields on which the recordset is sorted. Specify the name of each column in `IndexFieldNames` to use as an index for a table. Ordering of column names is significant. Separate names with semicolon. The specified columns don't need to be indexed. Set `IndexFieldNames` to an empty string to reset the recordset to the sort order originally used when the recordset's data was first retrieved.

Each field may optionally be followed by the keyword ASC / DESC or CIS / CS / BIN.

Use ASC, DESC keywords to specify a sort direction for the field. If one of these keywords is not used, the default sort direction for the field is ascending.

Use CIS, CS or BIN keywords to specify a sort type for string fields:
CIS - compare without case sensitivity;

CS - compare with case sensitivity;

BIN - compare by character ordinal values (this comparison is also case sensitive).

If a dataset uses a TCustomDAConnection component, the default value of sort type depends on the TCustomDAConnection.Options option of the connection. If a dataset does not use a connection (TVirtualTable dataset), the default is CS.

Read IndexFieldNames to determine the field (or fields) on which the recordset is sorted.

Ordering is processed locally.

**Note:** You cannot process ordering by BLOB fields. IndexFieldNames cannot be set to True when TCustomDADataset.UniDirectional=True.

**Example**

The following procedure illustrates how to set IndexFieldNames in response to a button click:

```pascal
DataSet1.IndexFieldNames := 'LastName ASC CIS; DateDue DESC';
```

5.13.1.1.2.3 KeyExclusive Property

Specifies the upper and lower boundaries for a range.

**Class**

TMemDataSet

**Syntax**

```pascal
property KeyExclusive: Boolean;
```

**Remarks**

Use KeyExclusive to specify whether a range includes or excludes the records that match its specified starting and ending values.

By default, KeyExclusive is False, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set KeyExclusive to True.
5.13.1.1.2.4 LocalConstraints Property

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Class

TMemDataSet

Syntax

```objectivec
property LocalConstraints: boolean default True;
```

Remarks

Use the LocalConstraints property to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet. When LocalConstraints is True, TMemDataSet ignores NOT NULL server constraints. It is useful for tables that have fields updated by triggers.

LocalConstraints is obsolete, and is only included for backward compatibility.

The default value is True.

5.13.1.1.2.5 LocalUpdate Property

Used to prevent implicit update of rows on database server.

Class

TMemDataSet

Syntax
**property** LocalUpdate: boolean **default** False;

Remarks
Set the LocalUpdate property to True to prevent implicit update of rows on database server. Data changes are cached locally in client memory.

© 1997-2021 Devart. All Rights Reserved.

### 5.13.1.1.2.6 Prepared Property

Determines whether a query is prepared for execution or not.

**Class**

**TMemDataSet**

**Syntax**

```plaintext
property Prepared: boolean;
```

**Remarks**
Determines whether a query is prepared for execution or not.

**See Also**
- **Prepare**

© 1997-2021 Devart. All Rights Reserved.

### 5.13.1.1.2.7 Ranged Property

Indicates whether a range is applied to a dataset.

**Class**

**TMemDataSet**

**Syntax**

```plaintext
property Ranged: Boolean;
```

**Remarks**

© 2021 Devart
Use the Ranged property to detect whether a range is applied to a dataset.

See Also
- SetRange
- SetRangeEnd
- SetRangeStart

Class
TMemDataSet

Syntax

```delphi
property UpdateRecordTypes: TUpdateRecordTypes default
[rtModified, rtInserted, rtUnmodified];
```

Remarks
Use the UpdateRecordTypes property to determine the update status for the current record when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateRecordTypes offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of records.

See Also
- CachedUpdates

Class
TMemDataSet

Syntax

```delphi
property UpdatesPending: Boolean default False;
```

Remarks
Used to check the status of the cached updates buffer.
TMemDataSet

Syntax

```plaintext
property UpdatesPending: boolean;
```

Remarks

Use the UpdatesPending property to check the status of the cached updates buffer. If UpdatesPending is True, then there are edited, deleted, or inserted records remaining in local cache and not yet applied to the database. If UpdatesPending is False, there are no such records in the cache.

See Also

- `CachedUpdates`

© 1997-2021 Devart. All Rights Reserved.

5.13.1.1.3 Methods

Methods of the TMemDataSet class.

For a complete list of the TMemDataSet class members, see the TMemDataSet Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyRange</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>CancelRange</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>Locate</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
</tbody>
</table>
### ApplyRange Method

Applies a range to the dataset.

#### Class

**TMemDataSet**

#### Syntax

```plaintext
procedure ApplyRange;
```

#### Remarks

Call `ApplyRange` to cause a range established with `SetRangeStart` and `SetRangeEnd` or `EditRangeStart` and `EditRangeEnd` to take effect.

When a range is in effect, only those records that fall within the range are available to the

---

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SetRangeStart</strong></td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong></td>
<td>Reads the status of the latest call to the <code>ApplyUpdates</code> method while cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdateStatus</strong></td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>
application for viewing and editing.

After a call to ApplyRange, the cursor is left on the first record in the range.

See Also
• CancelRange
• EditRangeEnd
• EditRangeStart
• IndexFieldNames
• SetRange
• SetRangeEnd
• SetRangeStart

5.13.1.1.3.2 ApplyUpdates Method

Writes dataset's pending cached updates to a database.

Class

TMemDataSet

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyUpdates</td>
<td>Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>ApplyUpdates(const UpdateRecKinds: TUpdateRecKinds)</td>
<td>Writes dataset's pending cached updates of specified records to a database.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.

© 2021 Devart
**procedure** ApplyUpdates; **overload; virtual**;

**Remarks**

Call the ApplyUpdates method to write a dataset’s pending cached updates to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database’s Rollback method to undo the changes if there is an error.

Following a successful write to the database, and following a successful call to a connection's Commit method, an application should call the CommitUpdates method to clear the cached update buffer.

**Note:** The preferred method for updating datasets is to call a connection component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The connection component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.

**Example**

The following procedure illustrates how to apply a dataset's cached updates to a database in response to a button click:

```pascal
procedure ApplyButtonClick(Sender: TObject);
begin
  with MyQuery do
  begin
    Session.StartTransaction;
    try
      ... <Modify data>
      ApplyUpdates; <try to write the updates to the database>
      Session.Commit; <on success, commit the changes>
    except
      RestoreUpdates; <restore update result for applied records>
      Session.Rollback; <on failure, undo the changes>
      raise; <raise the exception to prevent a call to CommitUpdates!>
    end;
    CommitUpdates; <on success, clear the cache>
  end;
end;
```

**See Also**

- `TMemDataSet.CachedUpdates`
- `TMemDataSet.CancelUpdates`
- `TMemDataSet.CommitUpdates`
**TMemDataSet.UpdateStatus**

Writes dataset's pending cached updates of specified records to a database.

**Class**

**TMemDataSet**

**Syntax**

```pascal
procedure ApplyUpdates(const UpdateRecKinds: TUpdateRecKinds);
```

**Parameters**

*UpdateRecKinds*

Indicates records for which the ApplyUpdates method will be performed.

**Remarks**

Call the ApplyUpdates method to write a dataset's pending cached updates of specified records to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database's Rollback method to undo the changes if there is an error.

Following a successful write to the database, and following a successful call to a connection's Commit method, an application should call the CommitUpdates method to clear the cached update buffer.

**Note:** The preferred method for updating datasets is to call a connection component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The connection component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.
5.13.1.3.3 CancelRange Method

Removes any ranges currently in effect for a dataset.

Class

TMemDataSet

Syntax

```plaintext
procedure CancelRange;
```

Remarks

Call CancelRange to remove a range currently applied to a dataset. Canceling a range reenables access to all records in the dataset.

See Also

- ApplyRange
- EditRangeEnd
- EditRangeStart
- IndexFieldNames
- SetRange
- SetRangeEnd
- SetRangeStart

© 1997-2021 Devart. All Rights Reserved.

5.13.1.3.4 CancelUpdates Method

Clears all pending cached updates from cache and restores dataset in its prior state.

Class

TMemDataSet

Syntax

```plaintext
procedure CancelUpdates;
```

Remarks

Call the CancelUpdates method to clear all pending cached updates from cache and restore
dataset in its prior state.

It restores the dataset to the state it was in when the table was opened, cached updates were last enabled, or updates were last successfully applied to the database.

When a dataset is closed, or the CachedUpdates property is set to False, CancelUpdates is called automatically.

See Also
- CachedUpdates
- TMemDataSet.ApplyUpdates
- UpdateStatus

5.13.1.3.5 CommitUpdates Method

Clears the cached updates buffer.

Class
TMemDataSet

Syntax

procedure CommitUpdates;

Remarks

Call the CommitUpdates method to clear the cached updates buffer after both a successful call to ApplyUpdates and a database component's Commit method. Clearing the cache after applying updates ensures that the cache is empty except for records that could not be processed and were skipped by the OnUpdateRecord or OnUpdateError event handlers. An application can attempt to modify the records still in cache.

CommitUpdates also checks whether there are pending updates in the dataset. And if there are, it calls ApplyUpdates.

Record modifications made after a call to CommitUpdates repopulate the cached update buffer and require a subsequent call to ApplyUpdates to move them to the database.

See Also
- CachedUpdates
5.13.1.3.6 DeferredPost Method

Makes permanent changes to the database server.

**Class**

**TMemDataSet**

**Syntax**

```plaintext
procedure DeferredPost;
```

**Remarks**

Call DeferredPost to make permanent changes to the database server while retaining dataset in its state whether it is dsEdit or dsInsert.

Explicit call to the Cancel method after DeferredPost has been applied does not abandon modifications to a dataset already fixed in database.

5.13.1.3.7 EditRangeEnd Method

Enables changing the ending value for an existing range.

**Class**

**TMemDataSet**

**Syntax**

```plaintext
procedure EditRangeEnd;
```

**Remarks**

Call EditRangeEnd to change the ending value for an existing range.

To specify an end range value, call FieldByName after calling EditRangeEnd.
After assigning a new ending value, call **ApplyRange** to activate the modified range.

**See Also**
- **ApplyRange**
- **CancelRange**
- **EditRangeStart**
- **IndexFieldNames**
- **SetRange**
- **SetRangeEnd**
- **SetRangeStart**

© 1997-2021 Devart. All Rights Reserved.

**5.13.1.3.8 EditRangeStart Method**

Enables changing the starting value for an existing range.

**Class**

**TMemDataSet**

**Syntax**

```plaintext
procedure EditRangeStart;
```

**Remarks**

Call **EditRangeStart** to change the starting value for an existing range.

To specify a start range value, call **FieldByName** after calling **EditRangeStart**.

After assigning a new ending value, call **ApplyRange** to activate the modified range.

**See Also**
- **ApplyRange**
- **CancelRange**
- **EditRangeEnd**
- **IndexFieldNames**
- **SetRange**
- **SetRangeEnd**
- **SetRangeStart**
GetBlob Method

Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Class

**TMemDataSet**

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetBlob(const FieldName: string)</td>
<td>Retrieves TBlob object for a field or current record when its name is known.</td>
</tr>
<tr>
<td>GetBlob(Field: TField)</td>
<td>Retrieves TBlob object for a field or current record when the field itself is known.</td>
</tr>
</tbody>
</table>

Syntax

```pascal
function GetBlob(Field: TField): TBlob; overload;
```

Parameters

- **Field**
  - Holds an existing TField object.

Return Value

- TBlob object that was retrieved.

Remarks

Call the GetBlob method to retrieve TBlob object for a field or current record when only its name or the field itself is known. FieldName is the name of an existing field. The field should
have MEMO or BLOB type.

Retrieves TBlob object for a field or current record when its name is known.

Class

**TMemDataSet**

Syntax

```
function GetBlob(const FieldName: string): TBlob; overload;
```

**Parameters**

- **FieldName**
  
  Holds the name of an existing field.

**Return Value**

TBlob object that was retrieved.

**Example**

```
PgQuery1.GetBlob('Comment').SaveToFile('Comment.txt');
```

**See Also**

- **TBlob**

5.13.1.1.3.10  Locate Method

Searches a dataset for a specific record and positions the cursor on it.

Class

**TMemDataSet**

**Overload List**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locate(const KeyFields: array of TField; const KeyValues: variant; Options:</td>
<td>Searches a dataset by the specified fields for a specific record and positions cursor</td>
</tr>
</tbody>
</table>
Locate(const KeyFields: string; const KeyValues: variant; Options: TLocateOptions)

Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

**Class**

**TMemDataSet**

**Syntax**

```pascal
function Locate(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateOptions): boolean;

reintroduce; overload;
```

**Parameters**

- **KeyFields**
  - Holds TField objects in which to search.

- **KeyValues**
  - Holds the variant that specifies the values to match in the key fields.

- **Options**
  - Holds additional search latitude when searching in string fields.

**Return Value**

- True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

© 1997-2021 Devart. All Rights Reserved. Request Support  DAC Forum  Provide Feedback

Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

**Class**

**TMemDataSet**

**Syntax**
function Locate(const KeyFields: string; const KeyValues: variant; Options: TLocateOptions): boolean; overload; override;

Parameters

KeyFields
Holds a semicolon-delimited list of field names in which to search.

KeyValues
Holds the variant that specifies the values to match in the key fields.

Options
Holds additional search latitude when searching in string fields.

Return Value
True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

Remarks

Call the Locate method to search a dataset for a specific record and position cursor on it.

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. An example is provided below.

Options is a set that optionally specifies additional search latitude when searching in string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then Locate allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

Locate returns True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

The Locate function works faster when dataset is locally sorted on the KeyFields fields. Local dataset sorting can be set with the TMemDataSet.IndexFieldNames property.

Example

An example of specifying multiple search values:

```
with CustTable do
  Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P', '408-431-1000']), [loPartialKey]);
```
5.13.1.3.11 LocateEx Method

Excludes features that don't need to be included to the `TMemDataSet.Locate` method of `TDataSet`.

### Class

`TMemDataSet`

### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LocateEx(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateExOptions)</code></td>
<td>Excludes features that don't need to be included to the <code>TMemDataSet.Locate</code> method of <code>TDataSet</code> by the specified fields.</td>
</tr>
<tr>
<td><code>LocateEx(const KeyFields: string; const KeyValues: variant; Options: TLocateExOptions)</code></td>
<td>Excludes features that don't need to be included to the <code>TMemDataSet.Locate</code> method of <code>TDataSet</code> by the specified field names.</td>
</tr>
</tbody>
</table>

Excludes features that don't need to be included to the `TMemDataSet.Locate` method of `TDataSet` by the specified fields.

### Class

`TMemDataSet`

### Syntax

```pascal
function LocateEx(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateExOptions): boolean; overload;
```

### Parameters
KeyFields
    Holds TField objects to search in.

KeyValues
    Holds the values of the fields to search for.

Options
    Holds additional search parameters which will be used by the LocateEx method.

Return Value
    True, if a matching record was found. Otherwise returns False.

Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified field names.

Class
    TMemDataSet

Syntax

    function LocateEx(const KeyFields: string; const KeyValues: variant; Options: TLocateExOptions): boolean; overload;

Parameters

KeyFields
    Holds the fields to search in.

KeyValues
    Holds the values of the fields to search for.

Options
    Holds additional search parameters which will be used by the LocateEx method.

Return Value
    True, if a matching record was found. Otherwise returns False.

Remarks

Call the LocateEx method when you need some features not to be included to the TMemDataSet.Locate method of TDataSet.

LocateEx returns True if it finds a matching record, and makes that record the current one. Otherwise LocateEx returns False.

The LocateEx function works faster when dataset is locally sorted on the KeyFields fields.
Local dataset sorting can be set with the `TMemDataSet.IndexFieldNames` property.

**Note:** Please add the MemData unit to the "uses" list to use the TLocalExOption enumeration.

**See Also**
- `TMemDataSet.IndexFieldNames`
- `TMemDataSet.Locate`

© 1997-2021
Devart. All Rights Reserved.

5.13.1.3.12 Prepare Method

Allocates resources and creates field components for a dataset.

**Class**

`TMemDataSet`

**Syntax**

```plaintext
procedure Prepare; virtual;
```

**Remarks**

Call the Prepare method to allocate resources and create field components for a dataset. To learn whether dataset is prepared or not use the Prepared property.

The UnPrepare method unprepares a query.

**Note:** When you change the text of a query at runtime, the query is automatically closed and unprepared.

**See Also**
- `Prepared`
- `UnPrepare`
5.13.1.1.3.13 RestoreUpdates Method

Marks all records in the cache of updates as unapplied.

Class

TMemDataSet

Syntax

procedure RestoreUpdates;

Remarks

Call the RestoreUpdates method to return the cache of updates to its state before calling ApplyUpdates. RestoreUpdates marks all records in the cache of updates as unapplied. It is useful when ApplyUpdates fails.

See Also

- CachedUpdates
- TMemDataSet.ApplyUpdates
- CancelUpdates
- UpdateStatus

© 1997-2021 Devart. All Rights Reserved.

5.13.1.1.3.14 RevertRecord Method

Cancels changes made to the current record when cached updates are enabled.

Class

TMemDataSet

Syntax

procedure RevertRecord;

Remarks

Call the RevertRecord method to undo changes made to the current record when cached updates are enabled.
5.13.1.3.15  SaveToXML Method

Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Class

TMemDataSet

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveToXML(Destination: TStream)</td>
<td>Saves the current dataset data to a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SaveToXML(const FileName: string)</td>
<td>Saves the current dataset data to a file in the XML format compatible with ADO format.</td>
</tr>
</tbody>
</table>

Saves the current dataset data to a stream in the XML format compatible with ADO format.

Class

TMemDataSet

Syntax

```
procedure SaveToXML(Destination: TStream); overload;
```

Parameters

Destination

Holds a TStream object.
Remarks

Call the SaveToXML method to save the current dataset data to a file or a stream in the XML format compatible with ADO format.

If the destination file already exists, it is overwritten. It remains open from the first call to SaveToXML until the dataset is closed. This file can be read by other applications while it is opened, but they cannot write to the file.

When saving data to a stream, a TStream object must be created and its position must be set in a preferable value.

See Also

- TVirtualTable.LoadFromFile
- TVirtualTable.LoadFromStream

Saves the current dataset data to a file in the XML format compatible with ADO format.

Class

TMemDataSet

Syntax

```pascal
procedure SaveToXML(const FileName: string); overload;
```

Parameters

- `FileName` holds the name of a destination file.

See Also

- TVirtualTable.LoadFromFile
- TVirtualTable.LoadFromStream
5.13.1.3.16 SetRange Method

Sets the starting and ending values of a range, and applies it.

Class

TMemDataSet

Syntax

```pascal
procedure SetRange(const StartValues: array of System.TVarRec;
const EndValues: array of System.TVarRec; StartExclusive: Boolean = False;
EndExclusive: Boolean = False);
```

Parameters

- **StartValues**
  Indicates the field values that designate the first record in the range. In C++,
  `StartValues_Size` is the index of the last value in the `StartValues` array.

- **EndValues**
  Indicates the field values that designate the last record in the range. In C++,
  `EndValues_Size` is the index of the last value in the `EndValues` array.

- **StartExclusive**
  Indicates the upper and lower boundaries of the start range.

- **EndExclusive**
  Indicates the upper and lower boundaries of the end range.

Remarks

Call SetRange to specify a range and apply it to the dataset. The new range replaces the
currently specified range, if any.

SetRange combines the functionality of SetRangeStart, SetRangeEnd, and ApplyRange in a
single procedure call. SetRange performs the following functions:

1. Puts the dataset into dsSetKey state.
2. Erases any previously specified starting range values and ending range values.
3. Sets the start and end range values.
4. Applies the range to the dataset.

After a call to SetRange, the cursor is left on the first record in the range.

If either StartValues or EndValues has fewer elements than the number of fields in the current
index, then the remaining entries are ignored when performing a search.

See Also
5.13.1.3.17  SetRangeEnd Method

Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

Class

TMemDataSet

Syntax

```
procedure SetRangeEnd;
```

Remarks

Call SetRangeEnd to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL.

Subsequent field assignments made with FieldByName specify the actual set of ending values for a range.

After assigning end-range values, call ApplyRange to activate the modified range.

See Also

- ApplyRange
- CancelRange
- EditRangeStart
- IndexFieldNames
- KeyExclusive
- SetRange
- SetRangeStart
5.13.1.3.18 SetRangeStart Method

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

Class

TMemDataSet

Syntax

procedure SetRangeStart;

Remarks

Call SetRangeStart to put the dataset into dsSetKey state, erase any previous start range values, and set them to NULL.

Subsequent field assignments to FieldByName specify the actual set of starting values for a range.

After assigning start-range values, call ApplyRange to activate the modified range.

See Also

• ApplyRange
• CancelRange
• EditRangeStart
• IndexFieldNames
• SetRange
• SetRangeEnd

5.13.1.3.19 UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class
**TMemDataSet**

**Syntax**

```objectivec
procedure UnPrepare; virtual;
```

**Remarks**

Call the UnPrepare method to free the resources allocated for a previously prepared query on the server and client sides.

>Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

**See Also**
- Prepare

**5.13.1.1.3.20 UpdateResult Method**

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

**Class**

*TMemDataSet*

**Syntax**

```objectivec
function UpdateResult: TUpdateAction;
```

**Return Value**

a value of the TUpdateAction enumeration.

**Remarks**

Call the UpdateResult method to read the status of the latest call to the ApplyUpdates method while cached updates are enabled. UpdateResult reflects updates made on the records that have been edited, inserted, or deleted.

UpdateResult works on the record by record basis and is applicable to the current record only.
5.13.1.3.21 UpdateStatus Method

Indicates the current update status for the dataset when cached updates are enabled.

Class

TMemDataSet

Syntax

```delphi
function UpdateStatus: TUpdateStatus; override;
```

Return Value

a value of the TUpdateStatus enumeration.

Remarks

Call the UpdateStatus method to determine the current update status for the dataset when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateStatus offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of the dataset.

See Also

- CachedUpdates

© 1997-2021 Devart. All Rights Reserved.

5.13.1.4 Events

Events of the TMemDataSet class.

For a complete list of the TMemDataSet class members, see the TMemDataSet Members topic.

Public
### OnUpdateError Event

Occurs when an exception is generated while cached updates are applied to a database.

#### Class

**TMemDataSet**

#### Syntax

```delphi
property OnUpdateError: TUpdateErrorEvent;
```

#### Remarks

Write the OnUpdateError event handler to respond to exceptions generated when cached updates are applied to a database.

E is a pointer to an EDatabaseError object from which application can extract an error message and the actual cause of the error condition. The OnUpdateError handler can use this information to determine how to respond to the error condition.

UpdateKind describes the type of update that generated the error.

UpdateAction indicates the action to take when the OnUpdateError handler exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateError can handle or correct the error, set UpdateAction to uaRetry before exiting the error handler.

The error handler can use the TField.OldValue and TField.NewValue properties to evaluate...
error conditions and set TField.Value to a new value to reapply. In this case, set
UpdateAction to uaRetry before exiting.

Note: If a call to ApplyUpdates raises an exception and ApplyUpdates is not called within the
context of a try...except block, an error message is displayed. If the OnUpdateError handler
cannot correct the error condition and leaves UpdateAction set to uaFail, the error message is
displayed twice. To prevent redisplay, set UpdateAction to uaAbort in the error handler.

See Also
- CachedUpdates

5.13.1.1.4.2 OnUpdateRecord Event

Occurs when a single update component can not handle the updates.

Class
- TMemDataSet

Syntax

```property OnUpdateRecord: TUpdateRecordEvent;```

Remarks

Write the OnUpdateRecord event handler to process updates that cannot be handled by a
single update component, such as implementation of cascading updates, insertions, or
deletions. This handler is also useful for applications that require additional control over
parameter substitution in update components.

UpdateKind describes the type of update to perform.

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On
entry into the handler, UpdateAction is always set to uaFail. If OnUpdateRecord is successful,
it should set UpdateAction to uaApplied before exiting.

See Also
- CachedUpdates
5.13.2 Variables

Variables in the MemDS unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoNotRaiseExceptionOnUAFail</td>
<td>An exception will be raised if the value of the UpdateAction parameter is uaFail.</td>
</tr>
<tr>
<td>SendDataSetChangeEventAfterOpen</td>
<td>The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.</td>
</tr>
</tbody>
</table>

5.13.2.1 DoNotRaiseExceptionOnUAFail Variable

An exception will be raised if the value of the UpdateAction parameter is uaFail.

Unit

MemDS

Syntax

```plaintext
DoNotRaiseExceptionOnUAFail: boolean = False;
```

Remarks

Starting with PgDAC, if the OnUpdateRecord event handler sets the UpdateAction parameter to uaFail, an exception is raised. The default value of UpdateAction is uaFail. So, the exception will be raised when the value of this parameter is left unchanged.

To restore the old behaviour, set DoNotRaiseExceptionOnUAFail to True.
5.13.2.2 **SendDataSetChangeEventAfterOpen Variable**

The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.

**Unit**

`MemDS`

**Syntax**

```plaintext
SendDataSetChangeEventAfterOpen: boolean = True;
```

**Remarks**

Starting with PgDAC, the DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. This problem appears only under Windows XP when visual styles are enabled.

To disable sending this event, change the value of this variable to False.

© 1997-2021 Devart. All Rights Reserved.

5.14 **PgAccess**

This unit contains main components of PgDAC.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TCustomPgDataSet</code></td>
<td>A base class defining functionality for the classes derived from it.</td>
</tr>
<tr>
<td><code>TCustomPgStoredProc</code></td>
<td>A component for accessing and executing stored procedures and functions.</td>
</tr>
<tr>
<td><code>TCustomPgTable</code></td>
<td>A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.</td>
</tr>
<tr>
<td><code>TCustomPgTimeStampField</code></td>
<td>A base class defining functionality for the classes derived from it.</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TPgConnection</td>
<td>Represents an open connection to a PostgreSQL database.</td>
</tr>
<tr>
<td>TPgConnectionOptions</td>
<td>This class allows setting up the behaviour of the TPgConnection class.</td>
</tr>
<tr>
<td>TPgConnectionSSLOptions</td>
<td>This class is used to set up the behaviour of the TPgConnection class.</td>
</tr>
<tr>
<td>TPgCursorField</td>
<td>A class providing access to the PostgreSQL cursor fields.</td>
</tr>
<tr>
<td>TPgDataSetOptions</td>
<td>This class allows setting up the behaviour of the TCustomPgDataSet class.</td>
</tr>
<tr>
<td>TPgDataSource</td>
<td>TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td>TPgDateField</td>
<td>A class providing access to the PostgreSQL date fields.</td>
</tr>
<tr>
<td>TPgEncryptor</td>
<td>The class that performs encrypting and decrypting of data.</td>
</tr>
<tr>
<td>TPgGeometricField</td>
<td>A class providing access to the PostgreSQL geometric fields.</td>
</tr>
<tr>
<td>TPgIntervalField</td>
<td>A class providing access to the PostgreSQL interval fields.</td>
</tr>
<tr>
<td>TPgLargeObject</td>
<td>A class providing support of PostgreSQL large objects.</td>
</tr>
<tr>
<td>TPgLargeObjectField</td>
<td>A class providing access to the PostgreSQL large object fields.</td>
</tr>
<tr>
<td>TPgMetaData</td>
<td>A component for obtaining metainformation about database objects from the server.</td>
</tr>
<tr>
<td>TPgParam</td>
<td>A class that is used to set the values of individual parameters passed with queries or stored</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TPgParams</td>
<td>Used to control TPgParam objects.</td>
</tr>
<tr>
<td>TPgQuery</td>
<td>A component for executing queries and operating record sets. It also provides flexible way to update data.</td>
</tr>
<tr>
<td>TPgSQL</td>
<td>A component for executing SQL statements and calling stored procedures on the database server.</td>
</tr>
<tr>
<td>TPgStoredProc</td>
<td>A component for accessing and executing stored procedures and functions.</td>
</tr>
<tr>
<td>TPgTable</td>
<td>A component for retrieving and updating data in a single table without writing SQL statements.</td>
</tr>
<tr>
<td>TPgTimeField</td>
<td>A class providing access to the PostgreSQL time fields.</td>
</tr>
<tr>
<td>TPgTimeStampField</td>
<td>A class providing access to the PostgreSQL timestamp fields.</td>
</tr>
<tr>
<td>TPgTransaction</td>
<td>A component for managing transactions in an application.</td>
</tr>
<tr>
<td>TPgUpdateSQL</td>
<td>Lets you tune update operations for the DataSet component.</td>
</tr>
</tbody>
</table>

**Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgNoticeEvent</td>
<td>This type is used for the TPgConnection.OnNotice event.</td>
</tr>
<tr>
<td>TPgNotificationEvent</td>
<td>This type is used for the TPgConnection.OnNotification event.</td>
</tr>
</tbody>
</table>

**Enumerations**
**TPgIsolationLevel**

Specifies the way the transactions containing database modifications are handled.

## Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgDACVersion</td>
<td>The version of PostgreSQL Data Access Components.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

## 5.14.1 Classes

Classes in the **PgAccess** unit.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomPgDataSet</td>
<td>A base class defining functionality for the classes derived from it.</td>
</tr>
<tr>
<td>TCustomPgStoredProc</td>
<td>A component for accessing and executing stored procedures and functions.</td>
</tr>
<tr>
<td>TCustomPgTable</td>
<td>A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.</td>
</tr>
<tr>
<td>TCustomPgTimeStampField</td>
<td>A base class defining functionality for the classes derived from it.</td>
</tr>
<tr>
<td>TPgConnection</td>
<td>Represents an open connection to a PostgreSQL database.</td>
</tr>
<tr>
<td>TPgConnectionOptions</td>
<td>This class allows setting up the behaviour of the TPgConnection class.</td>
</tr>
<tr>
<td>TPgConnectionSSLOptions</td>
<td>This class is used to set up the behaviour of the</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TPgConnection</td>
<td>TPgConnection class.</td>
</tr>
<tr>
<td>TPgCursorField</td>
<td>A class providing access to the PostgreSQL cursor fields.</td>
</tr>
<tr>
<td>TPgDataSetOptions</td>
<td>This class allows setting up the behaviour of the TCustomPgDataSet class.</td>
</tr>
<tr>
<td>TPgDataSource</td>
<td>TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td>TPgDateField</td>
<td>A class providing access to the PostgreSQL date fields.</td>
</tr>
<tr>
<td>TPgEncryptor</td>
<td>The class that performs encrypting and decrypting of data.</td>
</tr>
<tr>
<td>TPgGeometricField</td>
<td>A class providing access to the PostgreSQL geometric fields.</td>
</tr>
<tr>
<td>TPgIntervalField</td>
<td>A class providing access to the PostgreSQL interval fields.</td>
</tr>
<tr>
<td>TPgLargeObject</td>
<td>A class providing support of PostgreSQL large objects.</td>
</tr>
<tr>
<td>TPgLargeObjectField</td>
<td>A class providing access to the PostgreSQL large object fields.</td>
</tr>
<tr>
<td>TPgMetaData</td>
<td>A component for obtaining meta-information about database objects from the server.</td>
</tr>
<tr>
<td>TPgParam</td>
<td>A class that is used to set the values of individual parameters passed with queries or stored procedures.</td>
</tr>
<tr>
<td>TPgParams</td>
<td>Used to control TPgParam objects.</td>
</tr>
<tr>
<td>TPgQuery</td>
<td>A component for executing queries and operating record sets. It also provides flexible way to update data.</td>
</tr>
</tbody>
</table>
| TPgSQL                           | A component for executing
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgStoredProc</td>
<td>A component for accessing and executing stored procedures and functions.</td>
</tr>
<tr>
<td>TPgTable</td>
<td>A component for retrieving and updating data in a single table without writing SQL statements.</td>
</tr>
<tr>
<td>TPgTimeField</td>
<td>A class providing access to the PostgreSQL time fields.</td>
</tr>
<tr>
<td>TPgTimeStampField</td>
<td>A class providing access to the PostgreSQL timestamp fields.</td>
</tr>
<tr>
<td>TPgTransaction</td>
<td>A component for managing transactions in an application.</td>
</tr>
<tr>
<td>TPgUpdateSQL</td>
<td>Lets you tune update operations for the DataSet component.</td>
</tr>
</tbody>
</table>

5.14.1.1 TCustomPgDataSet Class

A base class defining functionality for the classes derived from it.

For a list of all members of this type, see TCustomPgDataSet members.

Unit

PgAccess

Syntax

```
TCustomPgDataSet = class(TCustomDADataSet);
```

Remarks

TCustomPgDataSet is a base dataset component that defines functionality for classes derived from it. Applications never use TCustomPgDataSet objects directly. Instead they use descendants of TCustomPgDataSet, such as TPgQuery and TPgTable that inherit its dataset-related properties and methods.
Inheritance Hierarchy

**TMemDataSet**
**TCustomDADataSet**
**TCustomPgDataSet**

© 1997-2021 Devart. All Rights Reserved.

5.14.1.1.1 Members

**TCustomPgDataSet** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td>Connection (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>Cursor</td>
<td>Used to fetch data from the REFCURSOR parameter or REFCURSOR field.</td>
</tr>
<tr>
<td>DataTypeMap (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td>Debug (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>DetailFields (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Disconnected (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to keep dataset opened after connection is</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DMLRefresh</td>
<td>Used to refresh record by RETURNING clause when insert or update is performed.</td>
</tr>
<tr>
<td>FetchAll</td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td>FetchRows</td>
<td>(inherited from TCustomDADataset) Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>(inherited from TCustomDADataset) Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>(inherited from TCustomDADataset) Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>(inherited from TMemDataSet) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>IsQuery</td>
<td>(inherited from TCustomDADataset) Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from TMemDataSet) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>(inherited from TCustomDADataset) Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>KeySequence</td>
<td>Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.</td>
</tr>
<tr>
<td>LastInsertOID</td>
<td>Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong>  (inherited from <strong>TMemDataSet</strong>)</td>
<td><strong>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</strong></td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong>  (inherited from <strong>TMemDataSet</strong>)</td>
<td><strong>Used to prevent implicit update of rows on database server.</strong></td>
</tr>
<tr>
<td><strong>MacroCount</strong>  (inherited from <strong>TCustomDADataset</strong>)</td>
<td><strong>Used to get the number of macros associated with the Macros property.</strong></td>
</tr>
<tr>
<td><strong>Macros</strong>  (inherited from <strong>TCustomDADataset</strong>)</td>
<td><strong>Makes it possible to change SQL queries easily.</strong></td>
</tr>
<tr>
<td><strong>MasterFields</strong>  (inherited from <strong>TCustomDADataset</strong>)</td>
<td><strong>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</strong></td>
</tr>
<tr>
<td><strong>MasterSource</strong>  (inherited from <strong>TCustomDADataset</strong>)</td>
<td><strong>Used to specify the data source component which binds current dataset to the master one.</strong></td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td><strong>Used to specify the behaviour of TCustomPgDataSet object.</strong></td>
</tr>
<tr>
<td><strong>ParamCheck</strong>  (inherited from <strong>TCustomDADataset</strong>)</td>
<td><strong>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</strong></td>
</tr>
<tr>
<td><strong>ParamCount</strong>  (inherited from <strong>TCustomDADataset</strong>)</td>
<td><strong>Used to indicate how many parameters are there in the Params property.</strong></td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td><strong>Contains parameters for a query's SQL statement.</strong></td>
</tr>
<tr>
<td><strong>Prepared</strong>  (inherited from <strong>TMemDataSet</strong>)</td>
<td><strong>Determines whether a query is prepared for execution or not.</strong></td>
</tr>
<tr>
<td><strong>Ranged</strong>  (inherited from <strong>TMemDataSet</strong>)</td>
<td><strong>Indicates whether a range is applied to a dataset.</strong></td>
</tr>
<tr>
<td><strong>ReadOnly</strong>  (inherited from <strong>TCustomDADataset</strong>)</td>
<td><strong>Used to prevent users from updating, inserting, or</strong></td>
</tr>
<tr>
<td>Field Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>(inherited from TCustomDADataSet) Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>(inherited from TCustomDADataSet) Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SequenceMode</td>
<td>Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>SQL</td>
<td>(inherited from TCustomDADataSet) Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>(inherited from TCustomDADataSet) Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>(inherited from TCustomDADataSet) Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock</td>
<td>(inherited from TCustomDADataSet) Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>(inherited from TCustomDADataSet) Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh</td>
<td>(inherited from TCustomDADataSet) Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>(inherited from TCustomDADataSet) Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>(inherited from TCustomDADataSet) Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject</td>
<td>Used to point to an update object.</td>
</tr>
</tbody>
</table>

PostgreSQL Data Access Components

© 2021 Devart
object component which provides SQL statements that perform updates of read-only datasets.

**UpdateRecordTypes** (inherited from `TMemDataSet`)
Used to indicate the update status for the current record when cached updates are enabled.

**UpdatesPending** (inherited from `TMemDataSet`)
Used to check the status of the cached updates buffer.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddWhere</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td><strong>ApplyRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td><strong>CreateProcCall</strong></td>
<td>Generates the stored procedure call.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td><strong>Fetched</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to find out whether <code>TCustomDADataset</code> has fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to learn whether <code>TCustomDADataset</code> is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to learn whether <code>TCustomDADataset</code> is fetching all rows to the end.</td>
</tr>
<tr>
<td><strong>FindKey</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindNearest</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>GetDataType</strong></td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><strong>GetFieldObject</strong></td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td><strong>GetFieldPrecision</strong></td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td><strong>GetFieldScale</strong></td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong></td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td><strong>GetOrderBy</strong></td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td><strong>GetPgCursor</strong></td>
<td>Retrieves a TPgCursor object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgDate</strong></td>
<td>Retrieves a TPgDate object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgInterval</strong></td>
<td>Retrieves a TPgInterval object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgLargeObject</strong></td>
<td>Retrieves a TPgLargeObject object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgRow</strong></td>
<td>Retrieves a TPgRow object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgTime</strong></td>
<td>Retrieves a TPgTime object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgTimeStamp</strong></td>
<td>Retrieves a TPgTimeStamp object for a field with known name.</td>
</tr>
<tr>
<td><strong>GotoCurrent</strong></td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td><strong>Locate</strong></td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong></td>
<td>Overloaded. Excludes features that don't need to be included to the</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Lock</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td><strong>MacroByName</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>OpenNext</strong></td>
<td>Opens the next REFCURSOR for stored procedure that returns more than one cursor.</td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td>Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>Prepare</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>RefreshRecord</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
</tbody>
</table>
SetRangeStart (inherited from **TMemDataSet**)  
Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

**SQLSaved** (inherited from **TCustomDADataSet**)  
Determines if the **SQL** property value was saved to the **BaseSQL** property.

**UnLock** (inherited from **TCustomDADataSet**)  
Releases a record lock.

**UnPrepare** (inherited from **TMemDataSet**)  
Frees the resources allocated for a previously prepared query on the server and client sides.

**UpdateResult** (inherited from **TMemDataSet**)  
Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

**UpdateStatus** (inherited from **TMemDataSet**)  
Indicates the current update status for the dataset when cached updates are enabled.

<table>
<thead>
<tr>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td><strong>AfterExecute</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
</tr>
<tr>
<td><strong>AfterFetch</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
</tr>
<tr>
<td><strong>AfterUpdateExecute</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
</tr>
<tr>
<td><strong>BeforeFetch</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
</tr>
<tr>
<td><strong>BeforeUpdateExecute</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
</tr>
</tbody>
</table>
| **OnUpdateError** (inherited from **TMemDataSet**) | Occurs when an exception is generated while cached updates are applied to a record.
### Properties of the TCustomPgDataSet class.

For a complete list of the TCustomPgDataSet class members, see the TCustomPgDataSet Members topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL (inherited from TCustomDADataset)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions (inherited from TCustomDADataset)</td>
<td>Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td>Connection (inherited from TCustomDADataset)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>Cursor</td>
<td>Used to fetch data from the REFCURSOR parameter or REFCURSOR field.</td>
</tr>
<tr>
<td>DataTypeMap (inherited from TCustomDADataset)</td>
<td>Used to set data type mapping rules.</td>
</tr>
<tr>
<td>Debug (inherited from TCustomDADataset)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>DetailFields (inherited from TCustomDADataset)</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Disconnected</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>DMLRefresh</td>
<td>Used to refresh record by RETURNING clause when insert or update is performed.</td>
</tr>
<tr>
<td>FetchAll</td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td>FetchRows</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>IsQuery</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from <strong>TMemDataSet</strong>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>KeySequence</td>
<td>Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.</td>
</tr>
<tr>
<td>LastInsertOID</td>
<td>Returns OID of the record inserted by the last query for</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to prevent explicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MacroCount</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>Macros</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>MasterSource</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Used to specify the behaviour of TCustomPgDataSet object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Contains parameters for a query's SQL statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to prevent users from</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SequenceMode</td>
<td>Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
</tbody>
</table>
### UpdateObject

Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.

### UpdateRecordTypes (inherited from TMemDataSet)

Used to indicate the update status for the current record when cached updates are enabled.

### UpdatesPending (inherited from TMemDataSet)

Used to check the status of the cached updates buffer.

#### See Also

- TCustomPgDataSet Class
- TCustomPgDataSet Class Members

---

5.14.1.2.1  Cursor Property

Used to fetch data from the REFCURSOR parameter or REFCURSOR field.

#### Class

TCustomPgDataSet

#### Syntax

```property
Cursor: TPgRefCursor;
```

#### Remarks

Use the Cursor property to fetch data from the REFCURSOR parameter or REFCURSOR field. You can assign the value of TPgParam.AsCursor or TPgCursorField.AsCursor to the Cursor property. After assigning you can open the dataset once.

#### Example

```pascal
PgQuery1.Cursor := PgSQL1.ParamByName('Cur').AsCursor;
PgQuery1.Open;
```

#### See Also

- TPgRefCursor
5.14.1.1.2.2  DMLRefresh Property

Used to refresh record by RETURNING clause when insert or update is performed.

Class

TCustomPgDataSet

Syntax

```property DMLRefresh: boolean;```

Remarks

Use the DMLRefresh property to refresh record by RETURNING clause when insert or update is performed.

The default value is False.

**Note:** When the DMLRefresh property is set to True, the value of `TCustomDADataSet.RefreshOptions` is ignored to avoid refetching field values from the server.

5.14.1.1.2.3  FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

TCustomPgDataSet

Syntax

```property FetchAll: boolean default True;```

Remarks
When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

Opening a dataset in FetchAll = False mode requires an active transaction. When the FetchAll property is False, the first call to TMemDataSet.Locate and TMemDataSet.LocateEx methods may take a lot of time to retrieve additional records to the client side.

See Also
- TCustomDADataset.FetchRows

5.14.1.2.4 KeySequence Property

Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.

Class
- TCustomPgDataSet

Syntax

```
property KeySequence: string;
```

Remarks

Use the KeySequence property to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.

Note: KeySequence is used by TCustomPgDataSet only if the KeyFields property is assigned.

See Also
- TCustomDADataset.KeyFields
- SequenceMode
5.14.1.1.2.5 LastInsertOID Property

Returns OID of the record inserted by the last query for table with OIDs.

Class

TCustomPgDataSet

Syntax

**property** LastInsertOID: Int64;

Remarks

Use the LastInsertOID property to get OID of the record inserted by the last query for table with OIDs.

5.14.1.1.2.6 Options Property

Used to specify the behaviour of TCustomPgDataSet object.

Class

TCustomPgDataSet

Syntax

**property** Options: T PgDataSetOptions;

Remarks

Set the properties of Options to specify the behaviour of a TCustomPgDataSet object.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoDeleteBlob</td>
<td>Used to delete large objects from the database automatically when a record is deleted from the dataset.</td>
</tr>
<tr>
<td>CacheBlobs</td>
<td>Used to allocate local memory buffer to hold a copy of the large object content.</td>
</tr>
<tr>
<td>CursorWithHold</td>
<td>Used to open query in the FetchAll=False mode without transaction.</td>
</tr>
<tr>
<td><strong>DeferredBlobRead</strong></td>
<td>Used to fetch values of large objects when they are explicitly requested.</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DistinctParams</strong></td>
<td>Used for correct TClientDataSet parameters handling.</td>
</tr>
<tr>
<td><strong>EnableBCD</strong></td>
<td>Used to enable currency type. Default value of this option is False.</td>
</tr>
<tr>
<td><strong>EnableFMTBCD</strong></td>
<td>Used to enable using FMTBCD instead of float for large integer numbers to keep precision.</td>
</tr>
<tr>
<td><strong>ExtendedFieldsInfo</strong></td>
<td>Used to perform an additional query to get information about returned fields and the tables they belong to.</td>
</tr>
<tr>
<td><strong>FullRefresh</strong></td>
<td>Used to specify the fields to include in the automatically generated SQL statement when calling the method.</td>
</tr>
<tr>
<td><strong>OIDAsInt</strong></td>
<td>Used to read OID fields as integer values and map these fields to TIntegerField.</td>
</tr>
<tr>
<td><strong>PrepareUpdateSQL</strong></td>
<td>Used to automatically prepare update queries before execution.</td>
</tr>
<tr>
<td><strong>SetEmptyStrToNull</strong></td>
<td>Force replace of empty strings with NULL values in data. The default value is False.</td>
</tr>
<tr>
<td><strong>UnknownAsString</strong></td>
<td>Used to map fields of unknown data types to TStringField (TWideStringField).</td>
</tr>
<tr>
<td><strong>UnpreparedExecute</strong></td>
<td>Used to apply simple executing to a SQL statement.</td>
</tr>
<tr>
<td><strong>UseParamTypes</strong></td>
<td>Used to disable automatic detection of the parameters data types.</td>
</tr>
</tbody>
</table>

Contains parameters for a query’s SQL statement.

**Class**

**TCustomPgDataSet**

**Syntax**

```plaintext
property Params: T PgParams stored False;
```

**Remarks**
Contains parameters for a query's SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information). Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

5.14.1.1.2.8 SequenceMode Property

Used to specify the methods used internally to generate a sequenced field.

Class

TCustomPgDataSet

Syntax

```property
SequenceMode: TSequenceMode default smPost;
```

Remarks

Set the SequenceMode property to specify which method is used internally to generate a sequenced field.

See Also

- TCustomDADataset.KeyFields
- KeySequence

5.14.1.2.9 UpdateObject Property

Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.

Class
**TCustomPgDataSet**

**Syntax**

```plaintext
property UpdateObject: TPgUpdateSQL;
```

**Remarks**

The UpdateObject property points to an update object component which provides SQL statements that perform updates of read-only datasets when cached updates are enabled.

© 1997-2021 Devart. All Rights Reserved.

© 2021 Devart

---

5.14.1.1.3 Methods

Methods of the `TCustomPgDataSet` class.

For a complete list of the `TCustomPgDataSet` class members, see the `TCustomPgDataSet Members` topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere (inherited from <code>TCustomDADataset</code>)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from <code>TMemDataSet</code>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>BreakExec (inherited from <code>TCustomDADataset</code>)</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange (inherited from <code>TMemDataSet</code>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates (inherited from <code>TMemDataSet</code>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates (inherited from <code>TMemDataSet</code>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td><strong>CreateProcCall</strong></td>
<td>Generates the stored procedure call.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>Execute</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td><strong>Fetched</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td><strong>FindKey</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindNearest</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>GetDataType</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><strong>GetFieldObject</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td><strong>GetFieldPrecision</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td><strong>GetFieldScale</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td><strong>GetOrderBy</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td><strong>GetPgCursor</strong></td>
<td>Retrieves a TPgCursor object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgDate</strong></td>
<td>Retrieves a TPgDate object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgInterval</strong></td>
<td>Retrieves a TPgInterval object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgLargeObject</strong></td>
<td>Retrieves a TPgLargeObject object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgRow</strong></td>
<td>Retrieves a TPgRow object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgTime</strong></td>
<td>Retrieves a TPgTime object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgTimeStamp</strong></td>
<td>Retrieves a TPgTimeStamp object for a field with known name.</td>
</tr>
<tr>
<td><strong>GotoCurrent</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Sets the current record in this dataset similar to the</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Locate (inherited from TMemDataSet)</td>
<td>current record in another dataset.</td>
</tr>
<tr>
<td>LocateEx (inherited from TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>Lock (inherited from TCustomDADataset)</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>MacroByName (inherited from TCustomDADataset)</td>
<td>Locks the current record.</td>
</tr>
<tr>
<td>OpenNext</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>Prepare (inherited from TCustomDADataset)</td>
<td>Opens the next REFCURSOR for stored procedure that returns more than one cursor.</td>
</tr>
<tr>
<td>RefreshRecord (inherited from TCustomDADataset)</td>
<td>Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td>RestoreSQL (inherited from TCustomDADataset)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>RestoreUpdates (inherited from TMemDataSet)</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td>RevertRecord (inherited from TMemDataSet)</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td>SaveSQL (inherited from TCustomDADataset)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>SaveToXML (inherited from TMemDataSet)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SetOrderBy (inherited from TCustomDADataset)</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td></td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td></td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SQLSaved</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td><strong>UnLock</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdateStatus</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

**See Also**
- **TCustomPgDataSet Class**
- **TCustomPgDataSet Class Members**
**TCustomPgDataSet**

**Syntax**

```pascal
procedure CreateProcCall(Name: string; Overload: integer = 0);
```

**Parameters**

- **Name**
  the name of the stored procedure.

- **Overload**
  the number of the overloaded procedure.

**Remarks**

Call the CreateProcCall method to assign SQL statement that calls the stored procedure to the SQL property and fill the Params property. The Overload parameter contains the number of the overloaded procedure. Retrieves information about the procedure parameters from PostgreSQL server. After calling CreateProcCall you can execute stored procedure by the Execute method.

**FindParam Method**

Searches for and returns a parameter with the specified name.

**Class**

**TCustomPgDataSet**

**Syntax**

```pascal
function FindParam(const Value: string): TPgParam;
```

**Parameters**

- **Value**
  Holds the stored procedure name.

**Return Value**

the parameter, if a match was found. Nil otherwise.

**Remarks**

Call the FindParam method to find a parameter with the name passed in the Name argument.
If a match was found, FindParam returns the parameter. Otherwise, it returns nil.

See Also
- TPgParam
- TPgSQL.ParamByName

5.14.1.1.3.3 GetPgCursor Method

Retrieves a TPgCursor object for a field with known name.

Class
TCustomPgDataSet

Syntax

```pascal
function GetPgCursor(const FieldName: string): TPgRefCursor;
```

Parameters

- **FieldName**
  
  Holds the name of an existing field.

Return Value

- a TPgCursor object for a field with known name.

Remarks

Call the GetPgCursor method to retrieve a TPgCursor object for a field when only its name is known. FieldName is the name of an existing field.
function GetPgDate(const FieldName: string): TPgDate;

Parameters
FieldName
Holds the name of an existing field.

Return Value
a TPgDate object for a field with known name.

Remarks
Call the GetPgDate method to retrieve a TPgDate object for a field when only its name is known. FieldName is the name of an existing field.

5.14.1.1.3.5 GetPgInterval Method

Retrieves a TPgInterval object for a field with known name.

Class
TCustomPgDataSet

Syntax
function GetPgInterval(const FieldName: string): TPgInterval;

Parameters
FieldName
Holds the name of an existing field.

Return Value
a TPgInterval object for a field with known name.

Remarks
Call the GetPgInterval method to retrieve a TPgInterval object for a field when only its name is known. FieldName is the name of an existing field.

See Also
- TPgInterval
- TPgParam.AsPgInterval
5.14.1.3.6 GetPgLargeObject Method

Retrieves a TPgLargeObject object for a field with known name.

Class
TCustomPgDataSet

Syntax

\begin{verbatim}
function GetPgLargeObject(const FieldName: string): TPgLargeObject;
\end{verbatim}

Parameters

\textit{FieldName}

Holds the name of an existing field.

Return Value

a TPgLargeObject object for a field with known name.

Remarks

Call the GetPgLargeObject method to retrieve a TPgLargeObject object for a field when only its name is known. FieldName is the name of an existing field.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.3.7 GetPgRow Method

Retrieves a TPgRow object for a field with known name.

Class
TCustomPgDataSet

Syntax

\begin{verbatim}
function GetPgRow(const FieldName: string): TPgRow;
\end{verbatim}

Parameters

\textit{FieldName}

Holds the name of an existing field.

Return Value

a TPgRow object for a field with known name.
Remarks

Call the GetPgRow method to retrieve a TPgRow object for a field when only its name is known. FieldName is the name of an existing field.

5.14.1.3.8 GetPgTime Method

Retrieves a TPgTime object for a field with known name.

Class

TCustomPgDataSet

Syntax

function GetPgTime(const FieldName: string): TPgTime;

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TPgTime object for a field with known name.

Remarks

Call the GetPgTime method to retrieve a TPgTime object for a field when only its name is known. FieldName is the name of an existing field.

5.14.1.3.9 GetPgTimeStamp Method

Retrieves a TPgTimeStamp object for a field with known name.

Class

TCustomPgDataSet

Syntax

function GetPgTimeStamp(const FieldName: string): TPgTimeStamp;
Parameters

FieldName
Holds the name of an existing field.

Return Value
a TPgTimeStamp object for a field with known name.

Remarks
Call the GetPgTimeSpan method to retrieve a TPgTimeStamp object for a field when only its name is known. FieldName is the name of an existing field.

Class
TCustomPgDataSet

Syntax

function OpenNext: boolean;

Return Value
True, if the next cursor is opened. False, if there are no more cursors.

Remarks
Call the OpenNext method to open next REFCURSOR for stored procedure that returns more than one cursor. When you execute a stored procedure that has REFCURSOR parameters, PgDAC opens a cursor from the first parameter automatically. The OpenNext function closes current cursor and opens a cursor from the next REFCURSOR parameter.
**TCustomPgDataSet**

**Syntax**

```pascal
function ParamByName(const Value: string): T PgParam;
```

**Parameters**

- **Value**
  - Holds the parameter name.

**Return Value**

- the parameter, if a match was found. Otherwise an exception is raised.

**Remarks**

Call the ParamByName method to find a parameter with the name passed in the Name argument.

If a match is found, ParamByName returns the parameter. Otherwise, an exception is raised.

**See Also**

- T PgParam
- T PgSQL.FindParam

© 1997-2021 Devart. All Rights Reserved.

5.14.1.2 **TCustomPgStoredProc Class**

A component for accessing and executing stored procedures and functions.

For a list of all members of this type, see TCustomPgStoredProc members.

**Unit**

PgAccess

**Syntax**

```pascal
TCustomPgStoredProc = class(TCustomPgDataSet);
```

**Remarks**

Use TPgStoredProc to access stored procedures on the database server.

You need only to define the StoredProcName property, and the SQL statement to call the
stored procedure will be generated automatically.

Use the Execute method at runtime to generate request that instructs server to execute procedure and PrepareSQL to describe parameters at run time

Inheritance Hierarchy

```
TMemDataSet
  TCustomDADataset
    TCustomPgDataSet
      TCustomPgStoredProc
```

© 1997-2021
Devar. All Rights Reserved.

5.14.1.2.1 Members

**TCustomPgStoredProc** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BaseSQL</strong> (inherited from TCustomDADataset)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong> (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong> (inherited from TCustomDADataset)</td>
<td>Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td><strong>Connection</strong> (inherited from TCustomDADataset)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>Cursor</strong> (inherited from TCustomPgDataSet)</td>
<td>Used to fetch data from the REFCURSOR parameter or REFCURSOR field.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong> (inherited from TCustomDADataset)</td>
<td>Used to set data type mapping rules.</td>
</tr>
<tr>
<td><strong>Debug</strong> (inherited from TCustomDADataset)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DetailFields</td>
<td>Used to specify the fields that correspond to the foreign key fields from</td>
</tr>
<tr>
<td></td>
<td>MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Disconnected</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>DMLRefresh</td>
<td>Used to refresh record by RETURNING clause when insert or update is</td>
</tr>
<tr>
<td></td>
<td>performed.</td>
</tr>
<tr>
<td>FetchAll</td>
<td>Defines whether to request all records of the query from database server</td>
</tr>
<tr>
<td></td>
<td>when the dataset is being opened.</td>
</tr>
<tr>
<td>FetchRows</td>
<td>Used to define the number of rows to be transferred across the network at</td>
</tr>
<tr>
<td></td>
<td>the same time.</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy,</td>
</tr>
<tr>
<td></td>
<td>and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>IsQuery</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate</td>
</tr>
<tr>
<td></td>
<td>properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>KeySequence</td>
<td>Used to specify the name of a sequence that will be used</td>
</tr>
<tr>
<td><strong>Field</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LastInsertOID</strong></td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MacroCount</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>Macros</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>MasterSource</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Used to specify the behaviour of TCustomPgDataSet object.</td>
</tr>
<tr>
<td><strong>Overload</strong></td>
<td>Used to specify the overloading number.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to indicate how many parameters are there in the Params property.</td>
</tr>
</tbody>
</table>
| **Params**               | (inherited from **TCustomPgDataSet**) Contains parameters for a
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from TMemDataSet) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>(inherited from TMemDataSet) Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>(inherited from TCustomDADataset) Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>(inherited from TCustomDADataset) Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>(inherited from TCustomDADataset) Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SequenceMode</strong></td>
<td>(inherited from TCustomPgDataSet) Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>(inherited from TCustomDADataset) Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td>(inherited from TCustomDADataset) Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td>(inherited from TCustomDADataset) Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td>(inherited from TCustomDADataset) Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td>(inherited from TCustomDADataset) Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong></td>
<td>(inherited from TCustomDADataset) Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td><strong>SQLUpdate</strong></td>
<td>(inherited from TCustomDADataset) Used to specify a SQL statement.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject</td>
<td>Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>BreakExec</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>CreateBlobStream</td>
<td>Used to obtain a stream for reading data from or writing</td>
</tr>
</tbody>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateProcCall</td>
<td>Generates the stored procedure call.</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>ExecProc</td>
<td>Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Fetched</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>FindParam</strong> (inherited from TCustomPgDataSet)</td>
<td>Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>GetDataType</strong> (inherited from TCustomDADataset)</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><strong>GetFieldObject</strong> (inherited from TCustomDADataset)</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td><strong>GetFieldPrecision</strong> (inherited from TCustomDADataset)</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td><strong>GetFieldScale</strong> (inherited from TCustomDADataset)</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong> (inherited from TCustomDADataset)</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td><strong>GetOrderBy</strong> (inherited from TCustomDADataset)</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td><strong>GetPgCursor</strong> (inherited from TCustomPgDataSet)</td>
<td>Retrieves a TPgCursor object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgDate</strong> (inherited from TCustomPgDataSet)</td>
<td>Retrieves a TPgDate object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgInterval</strong> (inherited from TCustomPgDataSet)</td>
<td>Retrieves a TPgInterval object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgLargeObject</strong> (inherited from TCustomPgDataSet)</td>
<td>Retrieves a TPgLargeObject object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgRow</strong> (inherited from TCustomPgDataSet)</td>
<td>Retrieves a TPgRow object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgTime</strong> (inherited from TCustomPgDataSet)</td>
<td>Retrieves a TPgTime object for a field with known name.</td>
</tr>
<tr>
<td><strong>GetPgTimeStamp</strong> (inherited from TCustomPgDataSet)</td>
<td>Retrieves a TPgTimeStamp object for a field with known name.</td>
</tr>
<tr>
<td><strong>GotoCurrent</strong> (inherited from TCustomDADataset)</td>
<td>Sets the current record in this dataset similar to the</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Locate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LocateEx</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Lock</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>MacroByName</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>OpenNext</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>ParamByName</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>Prepare</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>PrepareSQL</td>
<td></td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SaveSQL</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><strong>SetRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SQLSaved</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Determines if the <strong>SQL</strong> property value was saved to the <strong>BaseSQL</strong> property.</td>
</tr>
<tr>
<td><strong>UnLock</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdateStatus</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td><strong>AfterFetch</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td><strong>AfterUpdateExecute</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Occurs after executing insert, delete, update, lock</td>
</tr>
</tbody>
</table>
and refresh operations.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BaseSQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>Cursor</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to fetch data from the REFCURSOR parameter or REFCURSOR field.</td>
</tr>
<tr>
<td>Property</td>
<td>Class</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>DetailFields</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>Disconnected</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>DMLRefresh</strong></td>
<td><code>TCustomPgDataSet</code></td>
</tr>
<tr>
<td><strong>FetchAll</strong></td>
<td><code>TCustomPgDataSet</code></td>
</tr>
<tr>
<td><strong>FetchRows</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>FilterSQL</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>FinalSQL</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td><code>TMempDataSet</code></td>
</tr>
<tr>
<td><strong>IsQuery</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>KeyExclusive</strong></td>
<td><code>TMempDataSet</code></td>
</tr>
<tr>
<td><strong>KeyFields</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KeySequence</td>
<td>Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.</td>
</tr>
<tr>
<td>LastInsertOID</td>
<td>Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of TCustomPgDataSet object.</td>
</tr>
<tr>
<td>Overload</td>
<td>Used to specify the overloading number.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are generated.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>(inherited from <code>TCustomPgDataSet</code>) Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Contains parameters for a query's SQL statement.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SequenceMode</strong></td>
<td>(inherited from <code>TCustomPgDataSet</code>) Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
</tbody>
</table>
| **SQLRefresh**    | (inherited from `TCustomDADataset`) Used to specify a SQL statement that performs a database operation.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLUpdate</td>
<td>(inherited from TCustomDADataSet) Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>(inherited from TCustomDADataSet) Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject</td>
<td>(inherited from TCustomPgDataSet) Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>(inherited from TMemDataSet) Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending</td>
<td>(inherited from TMemDataSet) Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

See Also
- TCustomPgStoredProc Class
- TCustomPgStoredProc Class Members

© 1997-2021 Devart. All Rights Reserved.

5.14.1.2.2.1 Overload Property

Used to specify the overloading number.

Class

TCustomPgStoredProc

Syntax

```_PROPERTIES overload: integer default 0;```
Remarks

Set the Overload property to specify the overloading number in case the stored procedure is overloaded.

5.14.1.2.3  Methods

Methods of the TCustomPgStoredProc class.

For a complete list of the TCustomPgStoredProc class members, see the TCustomPgStoredProc Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere (inherited from TCustomDADataset)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>BreakExec (inherited from TCustomDADataset)</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange (inherited from TMemDataSet)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates (inherited from TMemDataSet)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates (inherited from TMemDataSet)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>CreateBlobStream (inherited from TCustomDADataset)</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CreateProcCall</td>
<td>(inherited from TCustomPgDataSet) Generates the stored procedure call.</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>(inherited from TMemDataSet) Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>(inherited from TCustomDADataSet) Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>(inherited from TMemDataSet) Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>(inherited from TMemDataSet) Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>ExecProc</td>
<td>(inherited from TCustomDADataSet) Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Execute</td>
<td>(inherited from TCustomDADataSet) Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>(inherited from TCustomDADataSet) Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Fetched</td>
<td>(inherited from TCustomDADataSet) Used to find out whether TCustomDADataSet has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>(inherited from TCustomDADataSet) Used to learn whether TCustomDADataSet is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>(inherited from TCustomDADataSet) Used to learn whether TCustomDADataSet is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>(inherited from TCustomDADataSet) Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>(inherited from TCustomDADataSet) Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>(inherited from TCustomDADataSet) Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td>FindParam</td>
<td>(inherited from TCustomPgDataSet) Searches for and returns a parameter with the specified...</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>--------</td>
<td>----------------</td>
</tr>
<tr>
<td>GetBlob</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>GetDataType</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>GetPgCursor</td>
<td>TCustomPgDataset</td>
</tr>
<tr>
<td>GetPgDate</td>
<td>TCustomPgDataset</td>
</tr>
<tr>
<td>GetPgInterval</td>
<td>TCustomPgDataset</td>
</tr>
<tr>
<td>GetPgLargeObject</td>
<td>TCustomPgDataset</td>
</tr>
<tr>
<td>GetPgRow</td>
<td>TCustomPgDataset</td>
</tr>
<tr>
<td>GetPgTime</td>
<td>TCustomPgDataset</td>
</tr>
<tr>
<td>GetPgTimeStamp</td>
<td>TCustomPgDataset</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Locate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LocateEx</strong></td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td><strong>Lock</strong></td>
<td>Locks the current record.</td>
</tr>
<tr>
<td><strong>MacroByName</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>OpenNext</strong></td>
<td>Opens the next REFCURSOR for stored procedure that returns more than one cursor.</td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td>Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td><strong>PrepareSQL</strong></td>
<td>Describes the parameters of a stored procedure.</td>
</tr>
<tr>
<td><strong>RefreshRecord</strong></td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td><strong>RestoreSQL</strong></td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong></td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong></td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong></td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
</tbody>
</table>
**SetRange** (inherited from **TMemDataSet**)
Sets the starting and ending values of a range, and applies it.

**SetRangeEnd** (inherited from **TMemDataSet**)
Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

**SetRangeStart** (inherited from **TMemDataSet**)
Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

**SQLSaved** (inherited from **TCustomDADataset**)
Determines if the SQL property value was saved to the BaseSQL property.

**UnLock** (inherited from **TCustomDADataset**)
Releases a record lock.

**UnPrepare** (inherited from **TMemDataSet**)
Frees the resources allocated for a previously prepared query on the server and client sides.

**UpdateResult** (inherited from **TMemDataSet**)
Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

**UpdateStatus** (inherited from **TMemDataSet**)
Indicates the current update status for the dataset when cached updates are enabled.

---

**See Also**
- **TCustomPgStoredProc Class**
- **TCustomPgStoredProc Class Members**

© 1997-2021 Devart. All Rights Reserved.

5.14.1.2.3.1 ExecProc Method

Executes a SQL statement on the server.

Class
**TCustomPgStoredProc**

### Syntax

```plaintext
procedure ExecProc;
```

### Remarks

Call the ExecProc method to execute a SQL statement on the server. The ExecProc method is similar to the `TCustomDADataSet.Execute` method. It is included for compatibility with `TStoredProc`.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.2.3.2 PrepareSQL Method

Describes the parameters of a stored procedure.

### Class

**TCustomPgStoredProc**

### Syntax

```plaintext
procedure PrepareSQL;
```

### Remarks

Call the PrepareSQL method to describe parameters of a stored procedure and assign SQL statement that calls the stored procedure to the SQL property. If necessary, the Execute method calls it automatically. You can define parameters at design time if Parameters Editor is opened.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.3 TCustomPgTable Class

A base class that defines functionality for descendant classes which access data in a single table without writing SQL statements.

For a list of all members of this type, see `TCustomPgTable` members.
**PgAccess**

**Syntax**

```delphi
TCustomPgTable = class(TCustomPgDataSet);
```

**Remarks**

TCustomPgTable implements functionality to access data in a table. Use TCustomPgTable properties and methods to gain direct access to records and fields in an underlying server database without writing SQL statements.

**Inheritance Hierarchy**

- **TMemDataSet**
  - **TCustomDADataSet**
    - **TCustomPgDataSet**
      - **TCustomPgTable**

**See Also**

- **TPgTable**
- **TPgQuery**
- **Master/Detail Relationships**

© 1997-2021 Devart. All Rights Reserved.

5.14.1.3.1 Members

**TCustomPgTable** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Connection</strong> (inherited from TCustomDADataSet)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>Cursor</strong> (inherited from TCustomPgDataSet)</td>
<td>Used to fetch data from the REFCursor parameter or REFCursor field.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong> (inherited from TCustomDADataSet)</td>
<td>Used to set data type mapping rules.</td>
</tr>
<tr>
<td><strong>Debug</strong> (inherited from TCustomDADataSet)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong> (inherited from TCustomDADataSet)</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong> (inherited from TCustomDADataSet)</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong> (inherited from TCustomPgDataSet)</td>
<td>Used to refresh record by RETURNING clause when insert or update is performed.</td>
</tr>
<tr>
<td><strong>FetchAll</strong> (inherited from TCustomPgDataSet)</td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td><strong>FetchRows</strong> (inherited from TCustomDADataSet)</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td><strong>FilterSQL</strong> (inherited from TCustomDADataSet)</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</strong> (inherited from TCustomDADataSet)</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong> (inherited from TMemDataSet)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsQuery</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>KeySequence</td>
<td>Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.</td>
</tr>
<tr>
<td>LastInsertOID</td>
<td>Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td>Limit</td>
<td>Used to set the number of rows retrieved from the query.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Used to specify the data source component which binds current dataset to the</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Offset</td>
<td>Used to allow retrieving data from the server starting from the specified row.</td>
</tr>
<tr>
<td>Options</td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Used to specify the behaviour of TCustomPgDataSet object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Contains parameters for a query's SQL statement.</td>
</tr>
<tr>
<td>Prepared</td>
<td>(inherited from <strong>TMemDataSet</strong>) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SequenceMode</td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>SQL</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the SQL statement that will be used</td>
</tr>
</tbody>
</table>
### PostgreSQL Data Access Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLLock (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount (inherited from TCustomDADataset)</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLUpdate (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>UniDirectional (inherited from TCustomDADataset)</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject (inherited from TCustomPgDataSet)</td>
<td>Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.</td>
</tr>
<tr>
<td>UpdateRecordTypes (inherited from TMemDataSet)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending (inherited from TMemDataSet)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere (inherited from TCustomDADataset)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>Method Name</td>
<td>Inherited From</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td>BreakExec</td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td>CancelRange</td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td>CancelUpdates</td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td>CommitUpdates</td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td>CreateBlobStream</td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td>CreateProcCall</td>
<td><strong>TCustomPgDataSet</strong></td>
</tr>
<tr>
<td>DeferredPost</td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td>DeleteWhere</td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td>EditRangeStart</td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td>Execute</td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td>Executing</td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td>Fetched</td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td>Fetching</td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>(inherited from TCustomDADataset) Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>(inherited from TCustomDADataset) Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>(inherited from TCustomDADataset) Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>(inherited from TCustomDADataset) Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td>FindParam</td>
<td>(inherited from TCustomPgDataSet) Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>(inherited from TMemDataSet) Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>GetDataType</td>
<td>(inherited from TCustomDADataset) Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>(inherited from TCustomDADataset) Returns a multireference shared object from field.</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>(inherited from TCustomDADataset) Retrieves the precision of a number field.</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>(inherited from TCustomDADataset) Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>(inherited from TCustomDADataset) Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>(inherited from TCustomDADataset) Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GetPgCursor</td>
<td>(inherited from TCustomPgDataSet) Retrieves a TPgCursor object for a field with known name.</td>
</tr>
<tr>
<td>GetPgDate</td>
<td>(inherited from TCustomPgDataSet) Retrieves a TPgDate object for a field with known name.</td>
</tr>
<tr>
<td>GetPgInterval</td>
<td>(inherited from TCustomPgDataSet) Retrieves a TPgInterval object.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>GetPgLargeObject</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GetPgRow</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GetPgTime</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GetPgTimeStamp</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Locate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LocateEx</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Lock</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>MacroByName</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>OpenNext</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>ParamByName</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>Prepare</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>RevertRecord</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveSQL</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SQLSaved</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td><strong>UnLock</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdateStatus</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>
Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute (inherited from TCustomDADataSet)</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch (inherited from TCustomDADataSet)</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute (inherited from TCustomDADataSet)</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch (inherited from TCustomDADataSet)</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td>BeforeUpdateExecute (inherited from TCustomDADataSet)</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td>OnUpdateError (inherited from TMemDataSet)</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord (inherited from TMemDataSet)</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

Properties of the TCustomPgTable class.

For a complete list of the TCustomPgTable class members, see the TCustomPgTable Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL (inherited from TCustomDADataSet)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates (inherited from TMemDataSet)</td>
<td>Used to enable or disable</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Conditions</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>Cursor</strong></td>
<td>(inherited from <code>TCustomPgData</code>set) Used to fetch data from the <code>REFCURSOR</code> parameter or <code>REFCURSOR</code> field.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to specify the fields that correspond to the foreign key fields from <code>MasterFields</code> when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong></td>
<td>(inherited from <code>TCustomPgData</code>set) Used to refresh record by <code>RETURNING</code> clause when insert or update is performed.</td>
</tr>
<tr>
<td><strong>FetchAll</strong></td>
<td>(inherited from <code>TCustomPgData</code>set) Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td><strong>FetchRows</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td><strong>FilterSQL</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</strong></td>
<td>(inherited from <code>TCustomDADataset</code>) Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with...</td>
</tr>
<tr>
<td>IndexFieldNames (inherited from <strong>TMemDataSet</strong>)</td>
<td>expanded macros.</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>IsQuery</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyFields</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeySequence</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td><strong>LastInsertOID</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.</td>
</tr>
<tr>
<td><strong>Limit</strong></td>
<td>Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to set the number of rows retrieved from the query.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>MacroCount</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>Macros</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>MasterFields</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td></td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>MasterSource</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Offset</strong></td>
<td>Used to allow retrieving data from the server starting from the specified row.</td>
</tr>
<tr>
<td><strong>Options</strong>  (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to specify the behaviour of TCustomPgDataSet object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong>   (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Contains parameters for a query's SQL statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong>   (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SequenceMode</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td><strong>SQL</strong>      (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a SQL</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SQLInsert</strong></td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong></td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong></td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong></td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the <code>TCustomDADataSet.RefreshRecord</code> procedure.</td>
</tr>
<tr>
<td><strong>SQLUpdate</strong></td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td><strong>UniDirectional</strong></td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td><strong>UpdateObject</strong></td>
<td>Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong></td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong></td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

See Also

- [TCustomPgTable Class](#)
- [TCustomPgTable Class Members](#)
5.14.1.3.2.1 Limit Property

Used to set the number of rows retrieved from the query.

Class

TCustomPgTable

Syntax

```plaintext
property Limit: integer default -1;
```

Remarks

Use the Limit property to set the number of rows retrieved from the query. If Limit is -1, all records will be obtained.

See Also

- Offset

5.14.1.3.2.2 Offset Property

Used to allow retrieving data from the server starting from the specified row.

Class

TCustomPgTable

Syntax

```plaintext
property Offset: integer default 0;
```

Remarks

Use the Offset property to allow retrieving data from the server starting from the specified row. The default value is 0.

See Also
5.14.1.4 **TCustomPgTimeStampField Class**

A base class defining functionality for the classes derived from it.

For a list of all members of this type, see **TCustomPgTimeStampField** members.

### Unit

**PgAccess**

### Syntax

```pascal
TCustomPgTimeStampField = class(TField);
```

### Remarks

TCustomPgTimeStampField is a base dataset component that defines functionality for classes derived from it. Applications never use TCustomPgTimeStampField objects directly. Instead they use descendants of TCustomPgTimeStampField

© 1997-2021
Devart. All Rights Reserved.

### Members

**TCustomPgTimeStampField** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsPgTimeStamp</td>
<td>Used to provide access to a <strong>TPgTimeStamp</strong> object.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.

### Properties

Properties of the **TCustomPgTimeStampField** class.

© 2021 Devart
For a complete list of the `TCustomPgTimeStampField` class members, see the `TCustomPgTimeStampField Members` topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AsPgTimeStamp</code></td>
<td>Used to provide access to a <code>TPgTimeStamp</code> object.</td>
</tr>
</tbody>
</table>

See Also

- `TCustomPgTimeStampField Class`
- `TCustomPgTimeStampField Class Members`

5.14.1.4.2.1 `AsPgTimeStamp` Property

Used to provide access to a `TPgTimeStamp` object.

Class

`TCustomPgTimeStampField`

Syntax

```
property AsPgTimeStamp: TCustomPgTimeStamp;
```

Remarks

Use the `AsTimeStamp` property to get access to a `TPgTimeStampl object which you can use for manipulations with timestamp value.

5.14.1.5 `TPgConnection Class`

Represents an open connection to a PostgreSQL database.

For a list of all members of this type, see `TPgConnection` members.
**PgAccess**

**Syntax**

```cpp
TPgConnection = class(TCustomDAConnection);
```

**Remarks**

The TPgConnection component is used to maintain connection to a PostgreSQL database. After setting the Username, Password, Server, and Database properties, you can establish a connection to the database by calling the Connect method or setting the Connected property to True. There are also many properties at the connection level that affect the default behavior of the queries executed within this connection.

Use this component in conjunction with TPgQuery, TPqTable, TPgStoredProc or other components for convenient interoperation with PostgreSQL database.

TPgConnection object represents a unique connection to PostgreSQL database.

**Note:** If the port differs from the default one (5432) please use the Port property to set it.

**Inheritance Hierarchy**

`TCustomDAConnection`  
`TPgConnection`

© 1997-2021 Devart. All Rights Reserved.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ConnectDialog</code></td>
<td>(inherited from <code>TCustomDAConnection</code>) Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td><code>ConnectionTimeout</code></td>
<td>Used to set the time to wait while trying to establish a connection before terminating the attempt and generating an error.</td>
</tr>
<tr>
<td><strong>ConnectString</strong> (inherited from <em>TCustomDAConnection</em>)</td>
<td>Used to specify the connection information, such as: UserName, Password, Server, etc.</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ConvertEOL</strong> (inherited from <em>TCustomDAConnection</em>)</td>
<td>Allows customizing line breaks in string fields and parameters.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Used to specify the name of the database to be used once a connection is open.</td>
</tr>
<tr>
<td><strong>InTransaction</strong> (inherited from <em>TCustomDAConnection</em>)</td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong> (inherited from <em>TCustomDAConnection</em>)</td>
<td>Specifies whether a login dialog appears immediately before opening a new connection.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Specifies the behaviour of the TPgConnectionOptions object.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Used to specify a password for a connection.</td>
</tr>
<tr>
<td><strong>Pooling</strong> (inherited from <em>TCustomDAConnection</em>)</td>
<td>Enables or disables using connection pool.</td>
</tr>
<tr>
<td><strong>PoolingOptions</strong> (inherited from <em>TCustomDAConnection</em>)</td>
<td>Specifies the behaviour of connection pool.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Used to specify the port for connection.</td>
</tr>
<tr>
<td><strong>ProcessID</strong></td>
<td>Returns the process ID (PID) of the backend server process handling this connection.</td>
</tr>
<tr>
<td><strong>ProtocolVersion</strong></td>
<td>Used to set the version of protocol for communication with PostgreSQL server.</td>
</tr>
<tr>
<td><strong>Schema</strong></td>
<td>Used to change the search path of the connection to the specified schema, or get the first value from the search path.</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>Contains the server name.</td>
</tr>
<tr>
<td><strong>ServerVersion</strong></td>
<td>Holds the PostgreSQL server version number.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ServerVersionFull</td>
<td>Holds full PostgreSQL server version including version number and some additional information.</td>
</tr>
<tr>
<td>SSLOptions</td>
<td>Used to set the properties required for protected SSL connection with the server.</td>
</tr>
<tr>
<td>Username</td>
<td>Contains username.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyUpdates (inherited from TCustomDACConnection)</td>
<td>Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td>BreakExec</td>
<td>Used to cancel a query that is currently being executed on the connection.</td>
</tr>
<tr>
<td>Commit (inherited from TCustomDACConnection)</td>
<td>Commits current transaction.</td>
</tr>
<tr>
<td>Connect (inherited from TCustomDACConnection)</td>
<td>Establishes a connection to the server.</td>
</tr>
<tr>
<td>CreateDataSet</td>
<td>Returns a new instance of TPgQuery class and associates it with the connection object.</td>
</tr>
<tr>
<td>CreateMetaData</td>
<td>Returns a new instance of the TPgMetaData class.</td>
</tr>
<tr>
<td>CreateSQL</td>
<td>Returns a new instance of TPgSQL class and associates it with the connection object.</td>
</tr>
<tr>
<td>CreateTransaction</td>
<td>Returns a new instance of the TPgTransaction class.</td>
</tr>
<tr>
<td>Disconnect (inherited from TCustomDACConnection)</td>
<td>Performs disconnect.</td>
</tr>
<tr>
<td>ExecProc (inherited from TCustomDACConnection)</td>
<td>Allows to execute stored procedure or function providing its name and parameters.</td>
</tr>
<tr>
<td>ExecProcEx (inherited from TCustomDACConnection)</td>
<td>Allows to execute a stored procedure or function.</td>
</tr>
<tr>
<td>ExecSQL (inherited from TCustomDACConnection)</td>
<td>Executes a SQL statement with parameters.</td>
</tr>
<tr>
<td>ExecSQLEx (inherited from TCustomDACConnection)</td>
<td>Executes any SQL.</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetDatabaseNames</td>
<td>Returns a database list from the server.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetRowType</td>
<td>Overloaded. Used to get a TPgRowType object for composite type.</td>
</tr>
<tr>
<td>GetStoredProcNames</td>
<td>Returns a list of stored procedures from the server.</td>
</tr>
<tr>
<td>GetTableNames</td>
<td>Provides a list of available tables names.</td>
</tr>
<tr>
<td>MonitorMessage</td>
<td>Sends a specified message through the TCustomDASQLMonitor component.</td>
</tr>
<tr>
<td>Ping</td>
<td>Used to check state of connection to the server.</td>
</tr>
<tr>
<td>ReleaseSavepoint</td>
<td>Releases the specified savepoint without affecting any work that has been performed after its creation.</td>
</tr>
<tr>
<td>RemoveFromPool</td>
<td>Marks the connection that should not be returned to the pool after disconnect.</td>
</tr>
<tr>
<td>Rollback</td>
<td>Discards all current data changes and ends transaction.</td>
</tr>
<tr>
<td>RollbackToSavepoint</td>
<td>Cancels all updates for the current transaction.</td>
</tr>
<tr>
<td>Savepoint</td>
<td>Defines a point in the transaction to which you can roll back later.</td>
</tr>
<tr>
<td>StartTransaction</td>
<td>Overloaded. Starts a new user transaction against the database server.</td>
</tr>
</tbody>
</table>

Events
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnectionLost (inherited from TCustomDAConnection)</td>
<td>This event occurs when connection was lost.</td>
</tr>
<tr>
<td>OnError (inherited from TCustomDAConnection)</td>
<td>This event occurs when an error has arisen in the connection.</td>
</tr>
<tr>
<td>OnNotice</td>
<td>Occurs when a message or notice is received from PostgreSQL server.</td>
</tr>
<tr>
<td>OnNotification</td>
<td>Occurs when an asynchronous notification is received from PostgreSQL server.</td>
</tr>
</tbody>
</table>

Properties of the **TPgConnection** class.

For a complete list of the **TPgConnection** class members, see the [TPgConnection Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectDialog (inherited from TCustomDAConnection)</td>
<td>Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td>ConnectString (inherited from TCustomDAConnection)</td>
<td>Used to specify the connection information, such as: UserName, Password, Server, etc.</td>
</tr>
<tr>
<td>ConvertEOL (inherited from TCustomDAConnection)</td>
<td>Allows customizing line breaks in string fields and parameters.</td>
</tr>
<tr>
<td>InTransaction (inherited from TCustomDAConnection)</td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td>LoginPrompt (inherited from TCustomDAConnection)</td>
<td>Specifies whether a login dialog appears immediately before opening a new connection.</td>
</tr>
</tbody>
</table>
Table:<br>Poolinng (inherited from TCustomDACConnection) | Enables or disables using connection pool.<br>PoolingOptions (inherited from TCustomDACConnection) | Specifies the behaviour of connection pool.<br>ProcessID | Returns the process ID (PID) of the backend server process handling this connection.<br>ServerVersion | Holds the PostgreSQL server version number.<br>ServerVersionFull | Holds full PostgreSQL server version including version number and some additional information.<br><br>Published<br><br| Name | Description |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionTimeout</td>
<td>Used to set the time to wait while trying to establish a connection before terminating the attempt and generating an error.</td>
<td></td>
</tr>
<tr>
<td>Database</td>
<td>Used to specify the name of the database to be used once a connection is open.</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>Specifies the behaviour of the TPgConnectionOptions object.</td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>Used to specify a password for a connection.</td>
<td></td>
</tr>
<tr>
<td>Port</td>
<td>Used to specify the port for connection.</td>
<td></td>
</tr>
<tr>
<td>ProtocolVersion</td>
<td>Used to set the version of protocol for communication with PostgreSQL server.</td>
<td></td>
</tr>
<tr>
<td>Schema</td>
<td>Used to change the search path of the connection to the specified schema, or get the first value from the search path.</td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>Contains the server name.</td>
<td></td>
</tr>
</tbody>
</table>
SSLOptions | Used to set the properties required for protected SSL connection with the server.
Username | Contains username.

See Also
- TPgConnection Class
- TPgConnection Class Members

5.14.1.5.2.1 ConnectionTimeout Property

Used to set the time to wait while trying to establish a connection before terminating the attempt and generating an error.

Class
TPgConnection

Syntax

```property
ConnectionTimeout: integer default
DefValConnectionTimeout;
```

Remarks

Use the ConnectionTimeout property to set the time to wait while trying to establish a connection before terminating the attempt and generating an error. The 0 value indicates no limit. The default value of ConnectionTimeout is 15 seconds.

5.14.1.5.2.2 Database Property

Used to specify the name of the database to be used once a connection is open.

Class
TPgConnection
### Database Property

Syntax

```plaintext
property Database: string;
```

Remarks

Use the Database property to specify the name of the database to be used once a connection is open. If the Database property is empty, it is assumed that the name of a database is identical to the user name.

See Also
- TCustomDAConnection.Server
- Port
- TCustomDAConnection.Username
- TCustomDAConnection.Password
- TCustomDAConnection.GetDatabaseNames

### Options Property

Syntax

```plaintext
property Options: TPGConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TPGConnectionOptions object.

Class

TPGConnection

Option Name | Description
---|---
ApplicationName | Defines the application name for connecting to the server. This name will be displayed in the pg_stat_activity view and included in CSV log entries. Only printable
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Charset</strong></td>
<td>Used to set the character set that PgDAC uses to read and write character data.</td>
</tr>
<tr>
<td><strong>EnableBCD</strong></td>
<td>Used to enable currency type. Default value of this option is False.</td>
</tr>
<tr>
<td><strong>EnableComposites</strong></td>
<td>Used to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.</td>
</tr>
<tr>
<td><strong>EnableDomains</strong></td>
<td>Used to map fields of domain types to TField with the base domain type.</td>
</tr>
<tr>
<td><strong>EnableFMTBCD</strong></td>
<td>Used to enable using FMTBCD instead of float for large integer numbers to keep precision.</td>
</tr>
<tr>
<td><strong>EnableGeometrics</strong></td>
<td>Used to map fields of geometric types to TPgGeometricField.</td>
</tr>
<tr>
<td><strong>EnablePgTimeStamps</strong></td>
<td>Used to map DATE, TIME, and TIMESTAMP fields to TPgDateField, TPgTimeField, and TPgTimeStampField accordingly.</td>
</tr>
<tr>
<td><strong>ImmediateNotices</strong></td>
<td>Sets the behavior of receiving notices from PostgreSQL.</td>
</tr>
<tr>
<td><strong>IPVersion</strong></td>
<td>Used to specify Internet Protocol version.</td>
</tr>
<tr>
<td><strong>MessagesCharset</strong></td>
<td>Used to set a character set for PostgreSQL error messages.</td>
</tr>
<tr>
<td><strong>UseUnicode</strong></td>
<td>Used to specify a value indicating whether the UTF8 charset will be used.</td>
</tr>
</tbody>
</table>

5.14.1.5.2.4 **Password Property**

Used to specify a password for a connection.

**Class**

TPgConnection

**Syntax**

```property** Password: **string**;```
Remarks
Use the Password property to specify a password for a connection. When property is being changed TPgConnection calls Disconnect method.

See Also
- Username
- Server
- TCustomDAConnection.Connect

5.14.1.5.2.5 Port Property

Used to specify the port for connection.

Class
TPgConnection

Syntax

| property | Port: integer default PgDefValPort; |

Remarks
Use the Port property to specify the port number of PostgreSQL database for connection.

See Also
- TCustomDAConnection.Server
- Database

5.14.1.5.2.6 ProcessID Property

Returns the process ID (PID) of the backend server process handling this connection.

Class
TPgConnection
5.14.1.5.2.7 ProtocolVersion Property

Used to set the version of protocol for communication with PostgreSQL server.

Class
TPgConnection

Syntax


Remarks
Set ProtocolVersion to pv20 to work with PostgreSQL server version 7.3 or older that don't support protocol version 3.0. Set ProtocolVersion to pv30 to enforce the protocol version 3.0. Set ProtocolVersion to pvAuto to automatically select between protocol versions depending on the specific query. The default value is pvAuto.

5.14.1.5.2.8 Schema Property

Used to change the search path of the connection to the specified schema, or get the first value from the search path.

Class
TPgConnection
Syntax

```
property Schema: string stored IsSchemaStored;
```

Remarks

Use the Schema property to change the search path of the connection to the specified schema, or get the first value from the search path.

Set the Schema property to change the search path of the connection to the specified schema. When connection is open, read the value of the property to get the name of the current schema (the first value from the current search path).

The TPgQuery, TPgStoredProc and other PgDAC components can implicitly use this property. Therefore you may have problems in case your search path contains several schemas and you use objects not from the current one.

Class

TPgConnection

Syntax

```
property Server: string;
```

Remarks

Use the Server property to supply server name to handle server's request for a login. When the property is being changed TPgConnection calls the Disconnect method.

See Also

- Username
- Password
- TCustomDAConnection.Connect

© 1997-2021
Devart. All Rights Reserved.
5.14.1.5.2.10  ServerVersion Property

Holds the PostgreSQL server version number.

Class

TPgConnection

Syntax

property ServerVersion: string;

Remarks

Read the ServerVersion property to get the PostgreSQL server version number in a string, for example '8.2.5'.

Works only when a TPgConnection instance is connected.

5.14.1.5.2.11  ServerVersionFull Property

Holds full PostgreSQL server version including version number and some additional information.

Class

TPgConnection

Syntax

property ServerVersionFull: string;

Remarks

Read the ServerVersionFull property to get full PostgreSQL server version including version number and some additional information.

Works only when a TPgConnection instance is connected.
5.14.1.5.2.12 SSLOptions Property

Used to set the properties required for protected SSL connection with the server.

Class

TPgConnection

Syntax

```
property SSLOptions: TPgConnectionSSLOptions;
```

Remarks

Use the SSLOptions property to set the properties required for protected SSL connection with the server.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Used to determine whether or with what priority an SSL connection will be negotiated with the server.</td>
</tr>
</tbody>
</table>

5.14.1.5.2.13 Username Property

Contains username.

Class

TPgConnection

Syntax

```
property Username: string;
```

Remarks

Use the Username property to supply a user name to handle server’s request for a login. When the property is being changed TPgConnection calls Disconnect method.

**Note:** The UserName property will be used as a default value for the Database and Schema parameters if they are empty.
See Also
- **Password**
- **Server**
- **TCustomDAConnection.Connect**

5.14.1.5.3 Methods

Methods of the **TPgConnection** class.

For a complete list of the **TPgConnection** class members, see the **TPgConnection Members** topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td><strong>BreakExec</strong></td>
<td>Used to cancel a query that is currently being executed on the connection.</td>
</tr>
<tr>
<td><strong>Commit</strong></td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Commits current transaction.</td>
</tr>
<tr>
<td><strong>Connect</strong></td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Establishes a connection to the server.</td>
</tr>
<tr>
<td><strong>CreateDataSet</strong></td>
<td>Returns a new instance of TPgQuery class and associates it with the connection object.</td>
</tr>
<tr>
<td><strong>CreateMetaData</strong></td>
<td>Returns a new instance of the TPgMetaData class.</td>
</tr>
<tr>
<td><strong>CreateSQL</strong></td>
<td>Returns a new instance of TPgSQL class and associates it with the connection object.</td>
</tr>
<tr>
<td><strong>CreateTransaction</strong></td>
<td>Returns a new instance of the TPgTransaction class.</td>
</tr>
<tr>
<td><strong>Disconnect</strong></td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Performs disconnect.</td>
</tr>
<tr>
<td><strong>ExecProc</strong></td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Allows to execute stored procedure or function providing its name and</td>
</tr>
</tbody>
</table>
**ExecProcEx** (inherited from **TCustomDAConnection**)
Allows to execute a stored procedure or function.

**ExecSQL** (inherited from **TCustomDAConnection**)
Executes a SQL statement with parameters.

**ExecSQLEx** (inherited from **TCustomDAConnection**)
Executes any SQL statement outside the TQuery or TSQL components.

**GetDatabaseNames** (inherited from **TCustomDAConnection**)
Returns a database list from the server.

**GetKeyFieldNames** (inherited from **TCustomDAConnection**)
Provides a list of available key field names.

**GetRowType**
Overloaded. Used to get a TPgRowType object for composite type.

**GetStoredProcNames** (inherited from **TCustomDAConnection**)
Returns a list of stored procedures from the server.

**GetTableNames** (inherited from **TCustomDAConnection**)
Provides a list of available tables names.

**MonitorMessage** (inherited from **TCustomDAConnection**)
Sends a specified message through the TCustomDASQLMonitor component.

**Ping** (inherited from **TCustomDAConnection**)
Used to check state of connection to the server.

**ReleaseSavepoint**
Releases the specified savepoint without affecting any work that has been performed after its creation.

**RemoveFromPool** (inherited from **TCustomDAConnection**)
Marks the connection that should not be returned to the pool after disconnect.

**Rollback** (inherited from **TCustomDAConnection**)
Discards all current data changes and ends transaction.

**RollbackToSavepoint**
 Cancels all updates for the current transaction.

**Savepoint**
Defines a point in the transaction to which you can rollback later.
### 5.14.1.5.3.1 BreakExec Method

Used to cancel a query that is currently being executed on the connection.

**Class**

**TPgConnection**

**Syntax**

```delphi
procedure BreakExec;
```

**Remarks**

Call the BreakExec method to cancel a query that is currently being executed on the connection. If you call the Execute method of TPgQuery, it will not return until the query is executed. So you need to call the BreakExec method from another thread to cancel the query.

### 5.14.1.5.3.2 CreateDataSet Method

Returns a new instance of TPgQuery class and associates it with the connection object.

**Class**

**TPgConnection**

**Syntax**

```delphi
function CreateDataSet(AOwner: TComponent = nil): TCustomDADataset; override;
```
### 5.14.1.5.3.3 CreateMetaData Method

Returns a new instance of the TPgMetaData class.

**Class**

TPgConnection

**Syntax**

```delphi
function CreateMetaData: TDAMetaData; override;
```

**Return Value**

a new instance of the TPgMetaData class.

**Remarks**

The CreateMetaData method returns a new instance of the TPgMetaData class and associates it with the connection object.

### 5.14.1.5.3.4 CreateSQL Method

Returns a new instance of TPgSQL class and associates it with the connection object.

**Class**

TPgConnection

**Syntax**

```delphi
function CreateSQL: TCustomDASQL; override;
```
Return Value

a new instance of TPgSQL class.

Remarks

Call the CreateSQL to return a new instance of TPgSQL class and associates it with the connection object.

Class

TPgConnection

Syntax

function CreateTransaction: TDATransaction; override;

Return Value

a new instance of the TPgTransaction class.

Remarks

The createTransaction method returns a new instance of the TPgTransaction class.

Class

TPgConnection

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetRowType(TypeOID: integer)</td>
<td>Used to get a TPgRowType object for composite type.</td>
</tr>
</tbody>
</table>
GetRowType(const Name: string)

- Used to get a TPgRowType object for composite type by its name.

Class

TPgConnection

Syntax

function GetRowType(TypeOID: integer): TPgRowType;

Parameters

TypeOID
OID of the composite type.

Return Value
a TPgRowType object.

Remarks

Use the GetRowType function to set the RowType property of TPgRow when you know the OID of the composite type. You can also use a TPgRowType object to get a list of composite type attributes.
**TypeName**
Name of the composite type.

**Return Value**
a TPgRowType object.

**Remarks**
Use the GetRowType function to set the RowType property of TPgRow when you know the name of the composite type. You can also use a TPgRowType object to get a list of composite type attributes.

5.14.1.5.3.7 ReleaseSavepoint Method

Releases the specified savepoint without affecting any work that has been performed after its creation.

**Class**
TPgConnection

**Syntax**

```plaintext
procedure ReleaseSavepoint(const Name: string);
```

**Parameters**
Name
Holds the savepoint name.

**Remarks**
Call the ReleaseSavepoint method to release the specified savepoint without affecting any work that has been performed after its creation.

**See Also**
- RollbackToSavepoint
- Savepoint

© 1997-2021 Devart. All Rights Reserved.
5.14.1.5.3.8 RollbackToSavepoint Method

Cancels all updates for the current transaction.

Class
TPgConnection

Syntax

```plaintext
procedure RollbackToSavepoint(const Name: string);
```

Parameters

Name
Holds the name identifying the last defined savepoint.

Remarks

Call the RollbackToSavepoint method to cancel all updates for the current transaction and restore its state up to the moment of the last defined savepoint.

See Also

- ReleaseSavepoint
- Savepoint

5.14.1.5.3.9 Savepoint Method

Defines a point in the transaction to which you can roll back later.

Class
TPgConnection

Syntax

```plaintext
procedure Savepoint(const Name: string);
```

Parameters

Name
Holds the name of the savepoint.

Remarks
Call the Savepoint method to define a point in the transaction to which you can roll back later. As the parameter, you can pass any valid name to identify the savepoint.

To roll back to the last savepoint call `RollbackToSavepoint`.

See Also
- `ReleaseSavepoint`
- `RollbackToSavepoint`

5.14.1.5.3.10 StartTransaction Method

Starts a new user transaction against the database server.

Class
- `TPgConnection`

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StartTransaction</strong></td>
<td>Starts a new user transaction against the database server.</td>
</tr>
<tr>
<td><strong>StartTransaction(IsolationLevel: TPgIsolationLevel; ReadOnly: boolean)</strong></td>
<td>Starts a new user transaction against the database server.</td>
</tr>
</tbody>
</table>

Remarks

StartTransaction is an overload method for `TCustomDAConnection.StartTransaction`. Call the
StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction property. If InTransaction is True, it indicates that a transaction is already in progress, a subsequent call to StartTransaction without first calling TCustomDACConnection.Commit or TCustomDACConnection.Rollback to end the current transaction raises EDatabaseError. Calling StartTransaction when connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes or Rollback to cancel them.

Starts a new user transaction against the database server.

Class

TPgConnection

Syntax

procedure StartTransaction(IsolationLevel: TPgIsolationLevel; ReadOnly: boolean = False); reintroduce; overload;

Parameters

IsolationLevel
- Specifies how the transactions containing database modifications are handled.

ReadOnly
- If True, read-only transaction is started that cannot modify data in the database.

Remarks

Call the StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction property. If InTransaction is True, it indicates that a transaction is already in progress, a subsequent call to StartTransaction without first calling TCustomDACConnection.Commit or TCustomDACConnection.Rollback to end the current transaction raises EDatabaseError. Calling StartTransaction when connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes or Rollback to cancel them.
5.14.1.5.4 Events

Events of the TPgConnection class.

For a complete list of the TPgConnection class members, see the TPgConnection Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnectionLost (inherited from TCustomDAConnection)</td>
<td>This event occurs when connection was lost.</td>
</tr>
<tr>
<td>OnError (inherited from TCustomDAConnection)</td>
<td>This event occurs when an error has arisen in the connection.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnNotice</td>
<td>Occurs when a message or notice is received from PostgreSQL server.</td>
</tr>
<tr>
<td>OnNotification</td>
<td>Occurs when an asynchronous notification is received from PostgreSQL server.</td>
</tr>
</tbody>
</table>

See Also
- TPgConnection Class
- TPgConnection Class Members

5.14.1.5.4.1 OnNotice Event

Occurs when a message or notice is received from PostgreSQL server.

Class
TPgConnection

Syntax

**property** OnNotice: **TPgNoticeEvent**;

Remarks

The OnNotice event occurs when PostgreSQL server sends a message or multiple messages with Severity lower than ERROR.

To set the message levels to be sent to the client, use the PostgreSQL client_min_messages configuration parameter. For more information refer to the PostgreSQL documentation.

5.14.1.5.4.2  OnNotification Event

Occurs when an asynchronous notification is received from PostgreSQL server.

Class

TPgConnection

Syntax

**property** OnNotification: **TPgNotificationEvent**;

Remarks

The OnNotification event occurs when an asynchronous notification is received from PostgreSQL server. Notification can be sent using NOTIFY command. To receive notifications use LISTEN command. To stop receiving notifications use UNLISTEN command. You can read more information about these commands in the PostgreSQL documentation.

**Note:** Notifications can be received only when a call to the server is performed (for example, on executing SQL statement). While the connection is idle, notifications are not received. To receive notifications while the connection is idle, use the TPgAlerter component.
5.14.1.6 TPgConnectionOptions Class

This class allows setting up the behaviour of the TPgConnection class.

For a list of all members of this type, see TPgConnectionOptions members.

Unit

PgAccess

Syntax

TPgConnectionOptions = class(TDAConnectionOptions);

Inheritance Hierarchy

TDAConnectionOptions
TPgConnectionOptions

5.14.1.6.1 Members

TPgConnectionOptions class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowImplicitConnect</td>
<td>Specifies whether to allow or not implicit connection opening.</td>
</tr>
<tr>
<td>ApplicationName</td>
<td>Defines the application name for connecting to the server. This name will be displayed in the pg_stat_activity view and included in CSV log entries. Only printable ASCII characters may be used in the ApplicationName value. Other characters will be replaced with question marks (?).</td>
</tr>
<tr>
<td>Charset</td>
<td>Used to set the character set that PgDAC uses to read...</td>
</tr>
<tr>
<td><strong>DefaultSortType</strong> (inherited from <code>TDACollectionOptions</code>)</td>
<td>Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the <code>TMemDataSet.IndexFieldNames</code> property of a dataset.</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DisconnectedMode</strong> (inherited from <code>TDACollectionOptions</code>)</td>
<td>Used to open a connection only when needed for performing a server call and closes after performing the operation.</td>
</tr>
<tr>
<td><strong>EnableBCD</strong></td>
<td>Used to enable currency type. Default value of this option is False.</td>
</tr>
<tr>
<td><strong>EnableComposites</strong></td>
<td>Used to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.</td>
</tr>
<tr>
<td><strong>EnableDomains</strong></td>
<td>Used to map fields of domain types to <code>TField</code> with the base domain type.</td>
</tr>
<tr>
<td><strong>EnableFMTBCD</strong></td>
<td>Used to enable using FMTBCD instead of float for large integer numbers to keep precision.</td>
</tr>
<tr>
<td><strong>EnableGeometrics</strong></td>
<td>Used to map fields of geometric types to <code>TPgGeometricField</code>.</td>
</tr>
<tr>
<td><strong>EnablePgTimeStamps</strong></td>
<td>Used to map <code>DATE</code>, <code>TIME</code>, and <code>TIMESTAMP</code> fields to <code>TPgDateField</code>, <code>TPgTimeField</code>, and <code>TPgTimeStampField</code> accordingly.</td>
</tr>
<tr>
<td><strong>ImmediateNotices</strong></td>
<td>Sets the behavior of receiving notices from PostgreSQL.</td>
</tr>
<tr>
<td><strong>IPVersion</strong></td>
<td>Used to specify Internet Protocol version.</td>
</tr>
</tbody>
</table>
| **KeepDesignConnected** (inherited from `TADatabaseOptions`) | Used to prevent an application from establishing and write character data.
### Properties of the `TPgConnectionOptions` class.

For a complete list of the `TPgConnectionOptions` class members, see the `TPgConnectionOptions Members` topic.

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DefaultSortType</code></td>
<td>Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the <code>TMemDataSet.IndexFieldNames</code> property of a dataset.</td>
</tr>
<tr>
<td><code>DisconnectedMode</code></td>
<td>Used to open a connection only when needed for performing a server call and closes after performing the operation.</td>
</tr>
<tr>
<td><code>KeepDesignConnected</code></td>
<td>Used to prevent an application from establishing a connection at the time of startup.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  

Request Support  
DAC Forum  
Provide Feedback
### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AllowImplicitConnect</strong></td>
<td>Specifies whether to allow or not implicit connection opening.</td>
</tr>
<tr>
<td><strong>ApplicationName</strong></td>
<td>Defines the application name for connecting to the server. This name will be displayed in the pg_stat_activity view and included in CSV log entries. Only printable ASCII characters may be used in the ApplicationName value. Other characters will be replaced with question marks (?).</td>
</tr>
<tr>
<td><strong>Charset</strong></td>
<td>Used to set the character set that PgDAC uses to read and write character data.</td>
</tr>
<tr>
<td><strong>EnableBCD</strong></td>
<td>Used to enable currency type. Default value of this option is False.</td>
</tr>
<tr>
<td><strong>EnableComposites</strong></td>
<td>Used to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.</td>
</tr>
<tr>
<td><strong>EnableDomains</strong></td>
<td>Used to map fields of domain types to TField with the base domain type.</td>
</tr>
<tr>
<td><strong>EnableFMTBCD</strong></td>
<td>Used to enable using FMTBCD instead of float for large integer numbers to keep precision.</td>
</tr>
</tbody>
</table>
### TPgConnectionOptions Class

#### Syntax

```
property ApplicationName: string;
```

#### Remarks

**ApplicationName Property**

Defines the application name for connecting to the server. This name will be displayed in the `pg_stat_activity` view and included in CSV log entries. Only printable ASCII characters may be used in the `ApplicationName` value. Other characters will be replaced with question marks (?).
5.14.1.6.2.2 Charset Property

Used to set the character set that PgDAC uses to read and write character data.

Class

TPgConnectionOptions

Syntax

property Charset: string;

Remarks

Use the Charset property to set the character set that PgDAC uses to read and write character data.

© 1997-2021
Devart. All Rights Reserved.

5.14.1.6.2.3 EnableBCD Property

Used to enable currency type. Default value of this option is False.

Class

TPgConnectionOptions

Syntax

property EnableBCD: boolean;

© 1997-2021
Devart. All Rights Reserved.

5.14.1.6.2.4 EnableComposites Property

Used to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.
Class

TPgConnectionOptions

Syntax

```plaintext
property EnableComposites: boolean default True;
```

Remarks

Use the EnableComposites property to detect fields of composite (ROW) type and create separate fields for each attribute of composite type.

When EnableComposites is set to True, PgDAC detects fields of composite type and creates separate fields for each attribute of composite type. If the ObjectView property is set to True, PgDAC also creates an instance of TADTField that is parent for all attributes fields.

When EnableComposites is False, fields of composite types are not detected and mapped to TStringField, TMemoField, TWideStringField, or TWideMemoField depending on the UnknownAsString and UseUnicode options.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.6.2.5 EnableDomains Property

Used to map fields of domain types to TField with the base domain type.

Class

TPgConnectionOptions

Syntax

```plaintext
property EnableDomains: boolean default True;
```

Remarks

Use the EnableDomains property to map fields of domain types to TField with base domain type.

When EnableDomains is set to True, the base data type of the domain is detected and the field with the domain data type is mapped to TField with the corresponding type. When EnableDomains is False, fields of domain types are mapped to TStringField, TMemoField, TWideStringField, or TWideMemoField depending on the UnknownAsString and UseUnicode options.
5.14.1.6.2.6 EnableFMTBCD Property

Used to enable using FMTBCD instead of float for large integer numbers to keep precision.

Class

TPgConnectionOptions

Syntax

```delphi
property EnableFMTBCD: boolean;
```

Remarks

Use the EnableFMTBCD property to enable using FMTBCD instead of float for large integer numbers to keep precision.

5.14.1.6.2.7 EnableGeometrics Property

Used to map fields of geometric types to TPgGeometricField.

Class

TPgConnectionOptions

Syntax

```delphi
property EnableGeometrics: boolean default True;
```

Remarks

Use the EnableGeometrics property to map fields of geometric types to TPgGeometricField.

When EnableGeometrics is set to True, fields with POINT, LSEG, BOX, PATH, POLYGON, and CIRCLE data type are mapped to TPgGeometricField. When EnableGeometrics is False, fields of geometric types are mapped to TStringField, TMemoField, TWideStringField, or TWideMemoField depending on the UnknownAsString and UseUnicode options.
5.14.1.6.2.8 EnablePgTimeStamps Property

Used to map DATE, TIME, and TIMESTAMP fields to TPgDateField, TPgTimeField, and TPgTimeStampField accordingly.

Class

TPgConnectionOptions

Syntax

property EnablePgTimeStamps: boolean default False;

Remarks

Use the EnablePgTimeStamp property to map DATE, TIME, and TIMESTAMP fields to TPgDateField, TPgTimeField, and TPgTimeStampField accordingly.

By default this option is set to False, and DATE, TIME, and TIMESTAMP fields are mapped to standard TDateField, TTimeField, and TDateTimeField. Set this option to True to work with values than can be represented by TDateTime variable. These values include dates before 01-01-0100 or after 12-31-9999 and special values INFINITY and -INFINITY that can hold TIMESTAMP field. You can work with such values using special PgDAC field types: TPgDateField, TPgTimeField, and TPgTimeStampField. TPgTimeField and TPgTimeStampField allows also getting timezone value for fields WITH TIMEZONE.

5.14.1.6.2.9 ImmediateNotices Property

Sets the behavior of receiving notices from PostgreSQL.

Class

TPgConnectionOptions

Syntax

property ImmediateNotices: boolean default False;
Remarks

If True, the TPgConnection.OnNotice event occurs immediately without waiting for the SQL query to be completed. The default value is False.

Class

TPgConnectionOptions

Syntax

property IPVersion: TIPVersion default DefValIPVersion;

Remarks

Use the IPVersion property to specify Internet Protocol Version.

- ivIPBoth - either Internet Protocol Version 6 (IPv6) or Version 4 (IPv4) is used.
- ivIPv4 - The default value. Internet Protocol Version 4 (IPv4) is used.
- ivIPv6 - Internet Protocol Version 6 (IPv6) is used.

Note: When the TIPVersion property is set to ivIPBoth, a connection attempt is made via IPv6 if it is enabled in the operating system settings. If the connection attempt fails, a new connection attempt is made via IPv4.

See Also

- TIPVersion

5.14.1.6.2.11 MessagesCharset Property

Used to set a character set for PostgreSQL error messages.

Class

TPgConnectionOptions
Syntax

```pascal
property MessagesCharset: string;
```

Remarks

If the property value is empty, the character set from PostgreSQL settings will be used.

5.14.1.6.2.12 UseUnicode Property

Used to specify a value indicating whether the UTF8 charset will be used.

Class

```pascal
TPgConnectionOptions
```

Syntax

```pascal
property UseUnicode: boolean default DefValUseUnicode;
```

Remarks

Use the UseUnicode property to set PostgreSQL client charset to UTF8 and converts client data according to this charset.

When a value is assigned to this property, TPgConnection is closed.

5.14.1.7 TPgConnectionSSLOptions Class

This class is used to set up the behaviour of the TPgConnection class.

For a list of all members of this type, see TPgConnectionSSLOptions members.

Unit

```pascal
PgAccess
```

Syntax

```pascal
TPgConnectionSSLOptions = class(TDACConnectionSSLOptions);
```
Inheritance Hierarchy

TDAConnectionSSLOptions
TPgConnectionSSLOptions

5.14.1.7.1 Members

TPgConnectionSSLOptions class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACert (inherited from TDAConnectionSSLOptions)</td>
<td>Holds the path to the certificate authority file.</td>
</tr>
<tr>
<td>Cert (inherited from TDAConnectionSSLOptions)</td>
<td>Holds the path to the client certificate.</td>
</tr>
<tr>
<td>CipherList (inherited from TDAConnectionSSLOptions)</td>
<td>Holds the list of allowed SSL ciphers.</td>
</tr>
<tr>
<td>Key (inherited from TDAConnectionSSLOptions)</td>
<td>Holds the path to the private client key.</td>
</tr>
<tr>
<td>Mode</td>
<td>Used to determine whether or with what priority an SSL connection will be negotiated with the server.</td>
</tr>
</tbody>
</table>

5.14.1.7.2 Properties

Properties of the TPgConnectionSSLOptions class.

For a complete list of the TPgConnectionSSLOptions class members, see the TPgConnectionSSLOptions Members topic.

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACert (inherited from TDAConnectionSSLOptions)</td>
<td>Holds the path to the certificate authority file.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Cert (inherited from TDAConnectionSSLOptions)</td>
<td>Holds the path to the client certificate.</td>
</tr>
<tr>
<td>CipherList (inherited from TDAConnectionSSLOptions)</td>
<td>Holds the list of allowed SSL ciphers.</td>
</tr>
<tr>
<td>Key (inherited from TDAConnectionSSLOptions)</td>
<td>Holds the path to the private client key.</td>
</tr>
<tr>
<td>Mode</td>
<td>Used to determine whether or with what priority an SSL connection will be negotiated with the server.</td>
</tr>
</tbody>
</table>

**See Also**
- TPgConnectionSSLOptions Class
- TPgConnectionSSLOptions Class Members

**5.14.1.7.2.1 Mode Property**

Used to determine whether or with what priority an SSL connection will be negotiated with the server.

**Class**
TPgConnectionSSLOptions

**Syntax**

```plaintext
property Mode: TSSLMode default DefValSSLMode;
```

**Remarks**

Use the Mode property to determine whether or with what priority an SSL connection will be negotiated with the server.

**5.14.1.8 TPgCursorField Class**

A class providing access to the PostgreSQL cursor fields.

For a list of all members of this type, see TPgCursorField members.
Unit

PgAccess

Syntax

TPgCursorField = class(TDACursorField);

Remarks

The TPgCursorField class provides access to the PostgreSQL cursor fields.

Inheritance Hierarchy

TDACursorField
  TPgCursorField

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsCursor</td>
<td>Used to provide access to a TPgCursor object.</td>
</tr>
</tbody>
</table>

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
5.14.1.8.2.1  AsCursor Property

Used to provide access to a TPGCursor object.

Class
TPgCursorField

Syntax
property AsCursor: TPgRefCursor;

Remarks
Use the AsCursor property to provide access to a TPGCursor object you can use for
manipulations with the cursor value.

5.14.1.9  TPgDataSetOptions Class

This class allows setting up the behaviour of the TCustomPgDataSet class.

For a list of all members of this type, see TPgDataSetOptions members.

Unit
PgAccess

Syntax
TPgDataSetOptions = class(TDADatasetOptions);
Inheritance Hierarchy

**TDADatasetOptions**
**TPgDataSetOptions**

© 1997-2021
Devart. All Rights Reserved.

5.14.1.9.1 Members

**TPgDataSetOptions** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoDeleteBlob</td>
<td>Used to delete large objects from the database automatically when a record is deleted from the dataset.</td>
</tr>
<tr>
<td>AutoPrepare (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to execute automatic TCustomDADataset.Prepare on the query execution.</td>
</tr>
<tr>
<td>CacheBlobs</td>
<td>Used to allocate local memory buffer to hold a copy of the large object content.</td>
</tr>
<tr>
<td>CacheCalcFields (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to enable caching of the TField.Calculated and TField.Lookup fields.</td>
</tr>
<tr>
<td>CompressBlobMode (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td>CursorWithHold</td>
<td>Used to open query in the FetchAll=False mode without transaction.</td>
</tr>
<tr>
<td>DefaultValues (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to request default values/expressions from the server and assign them to the DefaultExpression property.</td>
</tr>
<tr>
<td>DeferredBlobRead</td>
<td>Used to fetch values of large objects when they are explicitly requested.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DetailDelay</strong></td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td><strong>DistinctParams</strong></td>
<td>Used for correct TClientDataSet parameters handling.</td>
</tr>
<tr>
<td><strong>EnableBCD</strong></td>
<td>Used to enable currency type. Default value of this option is False.</td>
</tr>
<tr>
<td><strong>EnableFMTBCD</strong></td>
<td>Used to enable using FMTBCD instead of float for large integer numbers to keep precision.</td>
</tr>
<tr>
<td><strong>ExtendedFieldsInfo</strong></td>
<td>Used to perform an additional query to get information about returned fields and the tables they belong to.</td>
</tr>
<tr>
<td><strong>FieldsOrigin</strong></td>
<td>Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.</td>
</tr>
<tr>
<td><strong>FlatBuffers</strong></td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td><strong>FullRefresh</strong></td>
<td>Used to specify the fields to include in the automatically generated SQL statement when calling the method.</td>
</tr>
<tr>
<td><strong>InsertAllSetFields</strong></td>
<td>Used to include all set dataset fields in the generated INSERT statement</td>
</tr>
<tr>
<td><strong>LocalMasterDetail</strong></td>
<td>Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LongStrings</td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td>MasterFieldsNullable</td>
<td>Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).</td>
</tr>
<tr>
<td>NumberRange</td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.</td>
</tr>
<tr>
<td>OIDAsInt</td>
<td>Used to read OID fields as integer values and map these fields to TIntegerField.</td>
</tr>
<tr>
<td>PrepareUpdateSQL</td>
<td>Used to automatically prepare update queries before execution.</td>
</tr>
<tr>
<td>QueryRecCount</td>
<td>Used for TCustomDADataset to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used for TCustomDADataset to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td>RemoveOnRefresh</td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td>RequiredFields</td>
<td>Used for TCustomDADataset to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ReturnParams</td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
<tr>
<td>SetEmptyStrToNull</td>
<td>Force replace of empty strings with NULL values in data. The default value is False.</td>
</tr>
<tr>
<td>SetFieldsReadOnly</td>
<td>Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.</td>
</tr>
<tr>
<td>StrictUpdate</td>
<td>Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.</td>
</tr>
<tr>
<td>TrimFixedChar</td>
<td>Specifies whether to discard all trailing spaces in the string fields of a dataset.</td>
</tr>
<tr>
<td>UnknownAsString</td>
<td>Used to map fields of unknown data types to TStringField (TWideStringField).</td>
</tr>
<tr>
<td>UnpreparedExecute</td>
<td>Used to apply simple executing to a SQL statement.</td>
</tr>
<tr>
<td>UpdateAllFields</td>
<td>Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
<tr>
<td>UpdateBatchSize</td>
<td>Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.</td>
</tr>
<tr>
<td>UseParamTypes</td>
<td>Used to disable automatic detection of the parameters data types.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback
5.14.1.9.2 Properties

Properties of the **TPgDataSetOptions** class.

For a complete list of the **TPgDataSetOptions** class members, see the [TPgDataSetOptions Members] topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPrepare</td>
<td>Used to execute automatic TCustomDADataset.Prepare on the query execution.</td>
</tr>
<tr>
<td>CacheCalcFields</td>
<td>Used to enable caching of the TField Calculated and TField Lookup fields.</td>
</tr>
<tr>
<td>CompressBlobMode</td>
<td>Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td>DefaultValues</td>
<td>Used to request default values/expressions from the server and assign them to the DefaultExpression property.</td>
</tr>
<tr>
<td>DetailDelay</td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td>FieldsOrigin</td>
<td>Used for TCustomDADataset to fill the Origin property of the TField objects by appropriate value when opening a dataset.</td>
</tr>
<tr>
<td>FlatBuffers</td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td>InsertAllSetFields</td>
<td>Used to include all set dataset fields in the generated INSERT statement.</td>
</tr>
<tr>
<td>LocalMasterDetail</td>
<td>Used for TCustomDADataset to use</td>
</tr>
<tr>
<td><strong>LongStrings</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to represent string fields with the length that is greater than 255 as TStringField.</td>
</tr>
<tr>
<td><strong>MasterFieldsNullable</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).</td>
</tr>
<tr>
<td><strong>NumberRange</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.</td>
</tr>
<tr>
<td><strong>QueryRecCount</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td><strong>QuoteNames</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.</td>
</tr>
<tr>
<td><strong>RemoveOnRefresh</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used for a dataset to locally remove a record that can not be found on the server.</td>
</tr>
<tr>
<td><strong>RequiredFields</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.</td>
</tr>
<tr>
<td><strong>ReturnParams</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to return the new value of fields to dataset after insert or update.</td>
</tr>
</tbody>
</table>
### SetFieldsReadOnly (inherited from TDADatasetOptions)

Used for a dataset to set the `ReadOnly` property to `True` for all fields that do not belong to UpdatingTable or can not be updated.

### StrictUpdate (inherited from TDADatasetOptions)

Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.

### TrimFixedChar (inherited from TDADatasetOptions)

Specifies whether to discard all trailing spaces in the string fields of a dataset.

### UpdateAllFields (inherited from TDADatasetOptions)

Used to include all dataset fields in the generated UPDATE and INSERT statements.

### UpdateBatchSize (inherited from TDADatasetOptions)

Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoDeleteBlob</td>
<td>Used to delete large objects from the database automatically when a record is deleted from the dataset.</td>
</tr>
<tr>
<td>CacheBlobs</td>
<td>Used to allocate local memory buffer to hold a copy of the large object content.</td>
</tr>
<tr>
<td>CursorWithHold</td>
<td>Used to open query in the <code>FetchAll=False</code> mode without transaction.</td>
</tr>
<tr>
<td>DeferredBlobRead</td>
<td>Used to fetch values of large objects when they are explicitly requested.</td>
</tr>
<tr>
<td>DistinctParams</td>
<td>Used for correct TClientDataSet parameters</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>EnableBCD</strong></td>
<td>Used to enable currency type. Default value of this option is False.</td>
</tr>
<tr>
<td><strong>EnableFMTBCD</strong></td>
<td>Used to enable using FMTBCD instead of float for large integer numbers to keep precision.</td>
</tr>
<tr>
<td><strong>ExtendedFieldsInfo</strong></td>
<td>Used to perform an additional query to get information about returned fields and the tables they belong to.</td>
</tr>
<tr>
<td><strong>FullRefresh</strong></td>
<td>Used to specify the fields to include in the automatically generated SQL statement when calling the method.</td>
</tr>
<tr>
<td><strong>OIDAsInt</strong></td>
<td>Used to read OID fields as integer values and map these fields to TIntegerField.</td>
</tr>
<tr>
<td><strong>PrepareUpdateSQL</strong></td>
<td>Used to automatically prepare update queries before execution.</td>
</tr>
<tr>
<td><strong>SetEmptyStrToNull</strong></td>
<td>Force replace of empty strings with NULL values in data. The default value is False.</td>
</tr>
<tr>
<td><strong>UnknownAsString</strong></td>
<td>Used to map fields of unknown data types to TStringField (TWideStringField).</td>
</tr>
<tr>
<td><strong>UnpreparedExecute</strong></td>
<td>Used to apply simple executing to a SQL statement.</td>
</tr>
<tr>
<td><strong>UseParamTypes</strong></td>
<td>Used to disable automatic detection of the parameters data types.</td>
</tr>
</tbody>
</table>

See Also

- TPgDataSetOptions Class
- TPgDataSetOptions Class Members
5.14.1.9.2.1  AutoDeleteBlob Property

Used to delete large objects from the database automatically when a record is deleted from the dataset.

Class
TPgDataSetOptions

Syntax

```pascal
property AutoDeleteBlob: boolean default True;
```

Remarks
Use the AutoDeleteBlob property to delete large objects from the database automatically when a record is deleted from the dataset.

If True, the large objects are deleted from the database automatically when a record that holds OIDs of these large objects is deleted from the dataset.

5.14.1.9.2.2  CacheBlobs Property

Used to allocate local memory buffer to hold a copy of the large object content.

Class
TPgDataSetOptions

Syntax

```pascal
property CacheBlobs: boolean default True;
```

Remarks
Use the CacheBlobs property to allocate local memory buffer to hold a copy of the large object content.

If True, local memory buffer is allocated to hold a copy of the large object content.

If False, value of large object is being read and written directly from/to database without
memory buffer on the client. This can save memory on the client for very large values.

5.14.1.9.2.3 CursorWithHold Property

Used to open query in the FetchAll=False mode without transaction.

Class

TPgDataSetOptions

Syntax

```
property CursorWithHold: boolean default False;
```

Remarks

Use the CursorWithHold option to open query in the FetchAll=False mode without transaction.

When this option is False (default), an active transaction is required to open a query in the FetchAll=False mode. If there is no active transaction, PgDAC opens additional internal connection and starts transaction on this connection.

When this option is True, PgDAC uses DECLARE CURSOR ... WITH HOLD statement to open the query. In this case no active transaction is required but this may take additional server resources.

5.14.1.9.2.4 DeferredBlobRead Property

Used to fetch values of large objects when they are explicitly requested.

Class

TPgDataSetOptions

Syntax

```
property DeferredBlobRead: boolean default False;
```

Remarks

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback
Use the DeferredBlobRead property to fetch values of large objects when they are explicitly requested.

If True, values of large objects are only fetched when they are explicitly requested.

Otherwise all values of large objects are fetched when dataset is opened. This option has no sense when the CacheBlobs option is set to False.

5.14.1.9.2.5 DistinctParams Property

Used for correct TClientDataSet parameters handling.

Class

TPgDataSetOptions

Syntax

```property DistinctParams: boolean default True;```

5.14.1.9.2.6 EnableBCD Property

Used to enable currency type. Default value of this option is False.

Class

TPgDataSetOptions

Syntax

```property EnableBCD: boolean;```

5.14.1.9.2.7 EnableFMTBCD Property

Used to enable using FMTBCD instead of float for large integer numbers to keep precision.

Class
**TPgDataSetOptions**

**Syntax**

```cpp
property EnableFMTBCD: boolean;
```

**Remarks**

Use the EnableFMTBCD property to enable using FMTBCD instead of float for large integer numbers to keep precision.

---

5.14.1.9.2.8 ExtendedFieldsInfo Property

Used to perform an additional query to get information about returned fields and the tables they belong to.

**Class**

**TPgDataSetOptions**

**Syntax**

```cpp
property ExtendedFieldsInfo: boolean default True;
```

**Remarks**

Use the ExtendedFieldsInfo property to perform an additional query to get information about returned fields and the tables they belong to.

If True, an additional query is performed to get information about returned fields and the tables they belong to. This information includes NOT NULL attribute of the field, SEQUENCE linked to the field, and table name corresponding to the field. Table name information is needed to detect fields that belong to updating table, and set read-only attribute for all other fields returned by the query.

---

5.14.1.9.2.9 FullRefresh Property

Used to specify the fields to include in the automatically generated SQL statement when calling the method.
Class
TPgDataSetOptions

Syntax

```delphi
property FullRefresh: boolean;
```

Remarks

Use the FullRefresh property to specify what fields to include in the automatically generated SQL statement when calling the `TCustomDADataset.RefreshRecord` method. If the FullRefresh property is True, all fields from a query are included into SQL statement to refresh a single record. If FullRefresh is False, only fields from `TPgQuery.UpdatingTable` are included.

**Note:** If FullRefresh is True, the refresh of SQL statement for complex queries and views may be generated with errors. The default value is False.

Class
TPgDataSetOptions

Syntax

```delphi
property OIDAsInt: boolean default False;
```

Remarks

Use the OIDAsInt property to read OID fields as integer values and map these fields to TIntegerField.

By default dataset treats all fields with OID data type in the table (except main OID field of table created WITH OIDS option) as descriptors of large objects. PgDAC will try to read large objects with OIDs from query, and will fail if these OIDs are not large object descriptors.

Set these option to False to read OIDs as integer values and map OID fields to TIntegerField. Set this option to True to read OIDs as large objects descriptors, and then read the values of
the corresponding large objects. When OIDAsInt is True, OID fields are mapped on TBlobField.

5.14.1.9.2.11 PrepareUpdateSQL Property

Used to automatically prepare update queries before execution.

Class

TPgDataSetOptions

Syntax

```pascal
property PrepareUpdateSQL: boolean;
```

Remarks

If True, update queries are automatically prepared before executing.

5.14.1.9.2.12 SetEmptyStrToNull Property

Force replace of empty strings with NULL values in data. The default value is False.

Class

TPgDataSetOptions

Syntax

```pascal
property SetEmptyStrToNull: boolean;
```

5.14.1.9.2.13 UnknownAsString Property

Used to map fields of unknown data types to TStringField (TWideStringField).
**TPgDataSetOptions**

**Syntax**

```plaintext
property UnknownAsString: boolean default False;
```

**Remarks**

Use the UnknownAsString to map fields of unknown data types to TStringField (TWideStringField).

If False, fields of unknown data types (including geometric types and composite type when the EnableGeometrics and EnableComposites options of TPgConnection are set to False) are mapped to TMemoField or TWideMemoField depending on the value of the UseUnicode option.

Memo is used because maximum length of values from such fields is unknown.

If True, fields of unknown data types are mapped to TStringField or TWideStringField depending on the value of the UseUnicode option. Size of fields is set to 8192. Values larger than this size are truncated.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.9.2.14 UnpreparedExecute Property

Used to apply simple executing to a SQL statement.

**Class**

**TPgDataSetOptions**

**Syntax**

```plaintext
property UnpreparedExecute: boolean default False;
```

**Remarks**

If the UnpreparedExecute property is set to True, the simple execute is used for SQL statement. Statement is not prepared before execute. It allows to add multiple statements separated by semicolon to the SQL property.
5.14.1.9.2.15 UseParamTypes Property

Used to disable automatic detection of the parameters data types.

Class

TPgDataSetOptions

Syntax

property UseParamTypes: boolean default False;

Remarks

Set the UseParamTypes option to True to disable automatic detection of parameter types. When this option is True, data types of parameters are set basing on the DataType property. When this option is False, data types of the parameters are detected by server automatically.

5.14.1.10 TPgDataSource Class

TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.

For a list of all members of this type, see TPgDataSource members.

Unit

PgAccess

Syntax

TPgDataSource = class(TCRDataSource);

Remarks

TPgDataSource provides an interface between a PgDAC dataset components and data-aware controls on a form.

TPgDataSource inherits its functionality directly from the TDataSource component.

At design-time assign individual data-aware components' DataSource properties from their drop-down listboxes.
Inheritance Hierarchy

**TCRDataSource**

**TPgDataSource**

© 1997-2021

Devart. All Rights Reserved.

5.14.1.10.1 Members

**TPgDataSource** class overview.

© 1997-2021

Devart. All Rights Reserved.

5.14.1.11 Members

**TPgDateField** class overview.

© 1997-2021

Devart. All Rights Reserved.

5.14.1.11 TPgDateField Class

A class providing access to the PostgreSQL date fields.

For a list of all members of this type, see **TPgDateField** members.

Unit

**PgAccess**

Syntax

```pascal
TPgDateField = class(TCustomPgTimeStampField);
```

Remarks

The TPgDateField class provides access to the PostgreSQL date fields.

Inheritance Hierarchy

**TCustomPgTimeStampField**

**TPgDateField**

© 1997-2021

Devart. All Rights Reserved.

5.14.1.11.1 Members

**TPgDateField** class overview.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsPgDate</td>
<td>Used to provide access to a TPgDate object.</td>
</tr>
<tr>
<td>AsPgTimeStamp</td>
<td>Used to provide access to a TPgTimeobject.</td>
</tr>
<tr>
<td>(inherited from</td>
<td></td>
</tr>
<tr>
<td>TCustomPgTimeStampField)</td>
<td></td>
</tr>
</tbody>
</table>

See Also
- TPgDateField Class
- TPgDateField Class Members

5.14.1.11.2.1 AsPgDate Property

Used to provide access to a TPgDate object.

### Class

TPgDateField
Syntax

**property** AsPgDate: **TPgDate**;

Remarks

Use the AsPgDate property to provide access to a TPgDate object you can use for manipulations with the date value.

---

5.14.1.12 TPgEncryptor Class

The class that performs encrypting and decrypting of data.

For a list of all members of this type, see **TPgEncryptor** members.

Unit

**PgAccess**

Syntax

```plaintext
TPgEncryptor = class(TCREncryptor);
```

Inheritance Hierarchy

**TCREncryptor**

**TPgEncryptor**

---

5.14.1.12.1 Members

**TPgEncryptor** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataHeader</strong></td>
<td>(inherited from <strong>TCREncryptor</strong>) Specifies whether the additional information is stored with the encrypted data.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>EncryptionAlgorithm (inherited from TCREncryptor)</td>
<td>Specifies the algorithm of data encryption.</td>
</tr>
<tr>
<td>HashAlgorithm (inherited from TCREncryptor)</td>
<td>Specifies the algorithm of generating hash data.</td>
</tr>
<tr>
<td>InvalidHashAction (inherited from TCREncryptor)</td>
<td>Specifies the action to perform on data fetching when hash data is invalid.</td>
</tr>
<tr>
<td>Password (inherited from TCREncryptor)</td>
<td>Used to set a password that is used to generate a key for encryption.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetKey (inherited from TCREncryptor)</td>
<td>Sets a key, using which data is encrypted.</td>
</tr>
</tbody>
</table>

5.14.1.13 TPgGeometricField Class

A class providing access to the PostgreSQL geometric fields.

For a list of all members of this type, see TPgGeometricField members.

Unit

PgAccess

Syntax

TPgGeometricField = class(TField);

Remarks

The TPgGeometricField class provides access to the PostgreSQL geometric fields.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsPgGeometric</td>
<td>Used to provide access to a TPgGeometric object.</td>
</tr>
</tbody>
</table>

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsPgGeometric</td>
<td>Used to provide access to a TPgGeometric object.</td>
</tr>
</tbody>
</table>

### See Also

- TPgGeometricField Class
- TPgGeometricField Class Members

## AsPgGeometric Property

Used to provide access to a TPgGeometric object.

### Class

TPgGeometricField

### Syntax

```property
AsPgGeometric: TPgGeometric;
```

### Remarks
Use the AsPgGeometric property to provide access to a TPgGeometric object you can use for manipulations with the date value.

5.14.1.14 TPgIntervalField Class

A class providing access to the PostgreSQL interval fields.

For a list of all members of this type, see TPgIntervalField members.

Unit

PgAccess

Syntax

TPgIntervalField = class(TField);

Remarks

The TPgIntervalFields class provides access to the PostgreSQL interval fields.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsPgInterval</td>
<td>Used to provide access to a TPgInterval object.</td>
</tr>
</tbody>
</table>

5.14.1.14.1 Members

TPgIntervalField class overview.

Properties

Properties of the TPgIntervalField class.

For a complete list of the TPgIntervalField class members, see the TPgIntervalField
Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsPgInterval</td>
<td>Used to provide access to a TPgInterval object.</td>
</tr>
</tbody>
</table>

See Also

- TPgIntervalField Class
- TPgIntervalField Class Members

5.14.1.14.2.1 AsPgInterval Property

Used to provide access to a TPgInterval object.

Class

TPgIntervalField

Syntax

```property AsPgInterval: TPgInterval;```

Remarks

Use the AsPgInterval property to provide access to a TPgInterval object you can use for manipulations with the interval value.

5.14.1.15 TPgLargeObject Class

A class providing support of PostgreSQL large objects.

For a list of all members of this type, see TPgLargeObject members.

Unit

PgAccess
Syntax

```plaintext
TPgLargeObject = class(TPgSQLLargeObject);
```

Remarks

The TPgLargeObject class provides support of PostgreSQL large objects.

Inheritance Hierarchy

- TSharedObject
  - TBlob
    - TCompressedBlob
      - TPgSQLLargeObject
        - TPgLargeObject

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>(inherited from TBlob) Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td>AsWideString</td>
<td>(inherited from TBlob) Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>Cached</td>
<td>(inherited from TPgSQLLargeObject) Used to specify whether the content of large object should be stored in memory buffer on the client.</td>
</tr>
<tr>
<td>Compressed</td>
<td>(inherited from TCompressedBlob) Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td>CompressedSize</td>
<td>(inherited from TCompressedBlob) Used to indicate compressed size of the Blob data.</td>
</tr>
<tr>
<td>Connection</td>
<td>Connection used to read and write value of large object.</td>
</tr>
<tr>
<td>IsUnicode</td>
<td>(inherited from TBlob) Gives choice of making TBlob store and process</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OID (inherited from TPgSQLLargeObject)</td>
<td>Holds the OID of a large object.</td>
</tr>
<tr>
<td>RefCount (inherited from TSharedObject)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size (inherited from TBlob)</td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign (inherited from TBlob)</td>
<td>Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td>Clear (inherited from TBlob)</td>
<td>Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td>CloseObject (inherited from TPgSQLLargeObject)</td>
<td>Closes the large object that was previously opened by the OpenObject method.</td>
</tr>
<tr>
<td>CreateObject (inherited from TPgSQLLargeObject)</td>
<td>Creates a new large object database.</td>
</tr>
<tr>
<td>LoadFromFile (inherited from TBlob)</td>
<td>Loads the contents of a file into a TBlob object.</td>
</tr>
<tr>
<td>LoadFromStream (inherited from TBlob)</td>
<td>Copies the contents of a stream into the TBlob object.</td>
</tr>
<tr>
<td>OpenObject (inherited from TPgSQLLargeObject)</td>
<td>Opens the large object specified by the OID property.</td>
</tr>
<tr>
<td>Read (inherited from TBlob)</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlob.</td>
</tr>
<tr>
<td>ReadBlob (inherited from TPgSQLLargeObject)</td>
<td>Reads the content of the large object from the database.</td>
</tr>
<tr>
<td>Release (inherited from TSharedObject)</td>
<td>Decrements the reference count.</td>
</tr>
<tr>
<td>SaveToFile (inherited from TBlob)</td>
<td>Saves the contents of the TBlob object to a file.</td>
</tr>
</tbody>
</table>
SaveToStream (inherited from TBlob)

Copies the contents of a TBlob object to a stream.

Truncate (inherited from TBlob)

Sets new TBlob size and discards all data over it.

UnlinkObject (inherited from TPgSQLLargeObject)

Deletes the large object specified by the OID property from the database.

Write (inherited from TBlob)

Stores a raw sequence of bytes into a TBlob object.

WriteBlob (inherited from TPgSQLLargeObject)

Writes the content of the large object to the database.

5.14.1.15.2 Properties

Properties of the TPgLargeObject class.

For a complete list of the TPgLargeObject class members, see the TPgLargeObject Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString (inherited from TBlob)</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td>AsWideString (inherited from TBlob)</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>Cached (inherited from TPgSQLLargeObject)</td>
<td>Used to specify whether the content of large object should be stored in memory buffer on the client.</td>
</tr>
<tr>
<td>Compressed (inherited from TCompressedBlob)</td>
<td>Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td>CompressedSize (inherited from TCompressedBlob)</td>
<td>Used to indicate compressed size of the Blob data.</td>
</tr>
<tr>
<td>Connection</td>
<td>Connection used to read and write value of large object.</td>
</tr>
<tr>
<td>IsUnicode (inherited from TBlob)</td>
<td>Gives choice of making TBlob store and process data in Unicode format or</td>
</tr>
</tbody>
</table>
### OID (inherited from TPgSQLLargeObject)
Holds the OID of a large object.

### RefCount (inherited from TSharedObject)
Used to return the count of reference to a TSharedObject object.

### Size (inherited from TBlob)
Used to learn the size of the TBlob value in bytes.

**See Also**
- TPgLargeObject Class
- TPgLargeObject Class Members

**5.14.1.15.2.1 Connection Property**

Connection used to read and write value of large object.

**Class**
TPgLargeObject

**Syntax**

```
property Connection: TPgConnection;
```

**Remarks**
If you call methods that read or write large object value from/to database, set the Connection property to an appropriate TPgConnection objects. Otherwise these methods will fail.

**5.14.1.16 TPgLargeObjectField Class**

A class providing access to the PostgreSQL large object fields.

For a list of all members of this type, see TPgLargeObjectField members.


**PgAccess**

**Syntax**

```cpp
TPgLargeObjectField = class(TBlobField);
```

**Remarks**

The TPgLargeObjectFields class provides access to the PostgreSQL large object fields.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsLargeObject</td>
<td>Used to provide access to a TPgLargeObject object.</td>
</tr>
</tbody>
</table>

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsLargeObject</td>
<td>Used to provide access to a TPgLargeObject object.</td>
</tr>
</tbody>
</table>

**See Also**

- TPgLargeObjectField Class
- TPgLargeObjectField Class Members
5.14.1.16.2.1 AsLargeObject Property

Used to provide access to a TPgLargeObject object.

Class

TPgLargeObjectField

Syntax

```delphi
property AsLargeObject: TPgLargeObject;
```

Remarks

Use the AsLargeObject property to provide access to a TPgLargeObject object you can use for manipulations with the large object value.

5.14.1.17 TPgMetaData Class

A component for obtaining metainformation about database objects from the server.

For a list of all members of this type, see TPgMetaData members.

Unit

PgAccess

Syntax

```delphi
TPgMetaData = class(TDAMetaData);
```

Remarks

The TPgMetaData component is used to obtain metainformation from the server about objects in the database, such as tables, table columns, stored procedures, etc.

Inheritance Hierarchy

```delphi
TMemDataSet
   TDAMetaData
```

© 2021 Devart
TPgMetaData

See Also
- TCustomDADataset.Debug
- TCustomDASQL.Debug
- DBMonitor

© 1997-2021
Devart. All Rights Reserved.

5.14.1.17.1 Members

TPgMetaData class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>(inherited from TMemDataSet) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Connection</td>
<td>(inherited from TDAMetaData) Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>(inherited from TMemDataSet) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from TMemDataSet) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>(inherited from TMemDataSet) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>(inherited from TMemDataSet) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MetaDataKind</td>
<td>(inherited from TDAMetaData) Used to specify which kind of metainformation to show.</td>
</tr>
<tr>
<td>Prepared</td>
<td>(inherited from TMemDataSet) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>(inherited from TMemDataSet) Indicates whether a range is applied to a dataset.</td>
</tr>
</tbody>
</table>
Restrictions (inherited from **TDAMetaData**) | Used to provide one or more conditions restricting the list of objects to be described.
---|---
UpdateRecordTypes (inherited from **TMemDataSet**) | Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**) | Used to check the status of the cached updates buffer.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>GetMetaDataKinds</strong> (inherited from <strong>TDAMetaData</strong>)</td>
<td>Used to get values acceptable in the MetaDataKind property.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetRestrictions</td>
<td>(inherited from TDMetaData) Used to find out which restrictions are applicable to a certain MetaDataKind.</td>
</tr>
<tr>
<td>Locate</td>
<td>(inherited from TMemDataSet) Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>(inherited from TMemDataSet) Overloaded. Excludes features that don’t need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Prepare</td>
<td>(inherited from TMemDataSet) Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>(inherited from TMemDataSet) Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>(inherited from TMemDataSet) Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>(inherited from TMemDataSet) Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetRange</td>
<td>(inherited from TMemDataSet) Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>(inherited from TMemDataSet) Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>(inherited from TMemDataSet) Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>(inherited from TMemDataSet) Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
</tbody>
</table>
**UpdateResult** (inherited from **TMemDataSet**)  
Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

**UpdateStatus** (inherited from **TMemDataSet**)  
Indicates the current update status for the dataset when cached updates are enabled.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnUpdateError</strong></td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td><strong>OnUpdateRecord</strong></td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

### 5.14.1.18 TPgParam Class

A class that is used to set the values of individual parameters passed with queries or stored procedures.

For a list of all members of this type, see **TPgParam** members.

**Unit**  
**PgAccess**

**Syntax**

```plaintext
TPgParam = class(TDAParam);
```

**Remarks**

Use the properties of TPgParam to set the value of a parameter. Objects that use parameters create TPgParam objects to represent these parameters. For example, TPgParam objects are used by TPgSQL, TCustomPgDataSet.
TPgParam shares many properties with TField, as both describe the value of a field in a dataset. However, a TField object has several properties to describe the field binding, and how the field is displayed, edited, or calculated that are not needed in a TPgParam object. Conversely, TPgParam includes properties that indicate how the field value is passed as a parameter.

Inheritance Hierarchy

TDAParam
  TPgParam

See Also
- TCustomPgDataSet
- TPgSQL
- TPgParams

© 1997-2021
Devart. All Rights Reserved.

5.14.1.18.1 Members

TPgParam class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsBlob</td>
<td>Used to set and read the value of the BLOB parameter as string.</td>
</tr>
<tr>
<td>AsBlobRef</td>
<td>Used to set and read the value of the BLOB parameter as a TBlob object.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Used to assign the value for a float field to a parameter.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Used to assign the value for an integer field to the parameter.</td>
</tr>
<tr>
<td>AsLargeInt</td>
<td>Used to assign the value for a LargeInteger field to the parameter.</td>
</tr>
<tr>
<td>AsMemo</td>
<td>Used to assign the value for</td>
</tr>
</tbody>
</table>
### AsMemoRef (inherited from **TDAParam**)
- **Description:**
  a memo field to the parameter.

### AsPgDate
- **Description:**
  Used to specify the parameter value when it represents the date type.

### AsPgInterval
- **Description:**
  Used to specify the parameter value when it represents the INTERVAL type.

### AsPgTime
- **Description:**
  Used to specify the parameter value when it represents the time type.

### AsPgTimeStamp
- **Description:**
  Used to specify the parameter value when it represents the TimeStamp type.

### AsSQLTimeStamp (inherited from **TDAParam**)
- **Description:**
  Used to specify the value of the parameter when it represents a SQL timestamp field.

### AsString (inherited from **TDAParam**)
- **Description:**
  Used to assign the string value to the parameter.

### AsWideString (inherited from **TDAParam**)
- **Description:**
  Used to assign the Unicode string value to the parameter.

### DataType (inherited from **TDAParam**)
- **Description:**
  Indicates the data type of the parameter.

### IsNull (inherited from **TDAParam**)
- **Description:**
  Used to indicate whether the value assigned to a parameter is NULL.

### ParamType (inherited from **TDAParam**)
- **Description:**
  Used to indicate the type of use for a parameter.

### Size (inherited from **TDAParam**)
- **Description:**
  Specifies the size of a string type parameter.

### Value (inherited from **TDAParam**)
- **Description:**
  Used to represent the value of the parameter as Variant.

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AssignField (inherited from **TDAParam**) | Assigns field name and field value to a param.
---|---
AssignFieldValue (inherited from **TDAParam**) | Assigns the specified field properties and value to a parameter.
LoadFromFile (inherited from **TDAParam**) | Places the content of a specified file into a TDAParam object.
LoadFromStream (inherited from **TDAParam**) | Places the content from a stream into a TDAParam object.
SetBlobData (inherited from **TDAParam**) | Overloaded. Writes the data from a specified buffer to BLOB.

### Properties of the **TPgParam** class.

For a complete list of the **TPgParam** class members, see the [TPgParam Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsBlob</strong> (inherited from <strong>TDAParam</strong>)</td>
<td>Used to set and read the value of the BLOB parameter as string.</td>
</tr>
<tr>
<td><strong>AsBlobRef</strong> (inherited from <strong>TDAParam</strong>)</td>
<td>Used to set and read the value of the BLOB parameter as a TBlob object.</td>
</tr>
<tr>
<td><strong>AsFloat</strong> (inherited from <strong>TDAParam</strong>)</td>
<td>Used to assign the value for a float field to a parameter.</td>
</tr>
<tr>
<td><strong>AsInteger</strong> (inherited from <strong>TDAParam</strong>)</td>
<td>Used to assign the value for an integer field to the parameter.</td>
</tr>
<tr>
<td><strong>AsLargeInt</strong> (inherited from <strong>TDAParam</strong>)</td>
<td>Used to assign the value for a LargeInteger field to the parameter.</td>
</tr>
<tr>
<td><strong>AsMemo</strong> (inherited from <strong>TDAParam</strong>)</td>
<td>Used to assign the value for a memo field to the</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.14.1.18.2 Properties
### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataType</strong></td>
<td>Indicates the data type of the parameter.</td>
</tr>
<tr>
<td><strong>ParamType</strong></td>
<td>Used to indicate the type of use for a parameter.</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>Specifies the size of a string type parameter.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>Used to represent the value of the parameter as Variant.</td>
</tr>
</tbody>
</table>
5.14.1.18.2.1 AsPgDate Property

Used to specify the parameter value when it represents the date type.

Class
TPgParam

Syntax

```property` AsPgDate: `TPgDate`;```

Remarks
Use the AsPgDate property to specify the parameter value when it represents the date type.

5.14.1.18.2.2 AsPgInterval Property

Used to specify the parameter value when it represents the INTERVAL type.

Class
TPgParam

Syntax

```property` AsPgInterval: `TPgInterval`;```

Remarks
Use the AsPgInterval property to specify the parameter value when it represents the INTERVAL type.
5.14.1.18.2.3  AsPgTime Property

Used to specify the parameter value when it represents the time type.

Class

TPgParam

Syntax

property AsPgTime: TPgTime;

Remarks

Use the AsPgTime property to specify the parameter value when it represents the time type.

5.14.1.18.2.4  AsPgTimeStamp Property

Used to specify the parameter value when it represents the TimeStamp type.

Class

TPgParam

Syntax

property AsPgTimeStamp: TPgTimeStamp;

Remarks

Use the AsPgTimeStamp property to specify the parameter value when it represents the TimeStamp type.

5.14.1.19 TPgParams Class

Used to control TPgParam objects.

For a list of all members of this type, see TPgParams members.
Unit

PgAccess

Syntax

TPgParams = class(TDAParams);

Remarks

Use TPgParams to manage a list of TPgParam objects for an object that uses field parameters. For example, TPgStoredProc objects and TPgQuery objects use TPgParams objects to create and access their parameters.

Inheritance Hierarchy

TDAParams
  TPgParams

See Also

- TPgParam
- TCustomDASQL.Params
- TCustomDADataset.Params

© 1997-2021 Devart. All Rights Reserved.
5.14.1.19.2 Properties

Properties of the TPgParams class.

For a complete list of the TPgParams class members, see the TPgParams Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Used to iterate through all field parameters.</td>
</tr>
</tbody>
</table>

See Also
- TPgParams Class
- TPgParams Class Members

5.14.1.19.2.1 Items Property(Indexer)

Used to iterate through all field parameters.

Class
TPgParams

Syntax

```delphi
property Items[Index: integer]: TPgParam; default;
```

Parameters

- `Index`
  - Holds the index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all field parameters. Index identifies the index in the
range 0..Count - 1. Items can refer to a particular parameter by its index, but the 
TDAParams.ParamByName method is preferred to avoid depending on the order of the 
parameters.

5.14.1.20 TPgQuery Class

A component for executing queries and operating record sets. It also provides flexible way to 
update data.

For a list of all members of this type, see TPgQuery members.

Unit
PgAccess

Syntax

TPgQuery = class(TCustomPgDataSet);

Remarks

TPgQuery is a direct descendant of the TCustomPgDataSet component. It publishes most of 
its inherited properties and events so that they can be manipulated at design-time.

Use TPgQuery to perform fetching, insertion, deletion and update of record by dynamically 
generated SQL statements. TPgQuery provides automatic blocking of records, their checking 
before edit and refreshing after post. Set SQL, SQLInsert, SQLDelete, SQLRefresh, and 
SQLUpdate properties to define SQL statements for subsequent accesses to the database 
server. There is no restriction to their syntax, so any SQL statement is allowed. Usually you 
need to use INSERT, DELETE, and UPDATE statements but you also may use stored 
procedures in more diverse cases.

To modify records, you can specify KeyFields. If they are not specified, TPgQuery will retrieve 
primary keys for UpdatingTable from metadata. TPgQuery can automatically update only one 
table. Updating table is defined by the UpdatingTable property if this property is set. 
Otherwise, the table a field of which is the first field in the field list in the SELECT clause is 
used as an updating table.

The SQLInsert, SQLDelete, SQLUpdate, SQLRefresh properties support automatic binding of 
parameters which have identical names to fields captions. To retrieve the value of a field as it 
was before the operation use the field name with the 'OLD_' prefix. This is especially useful
when doing field comparisons in the WHERE clause of the statement. Use the 
**TCustomDADataset.BeforeUpdateExecute** event to assign the value to additional parameters 
and the **TCustomDADataset.AfterUpdateExecute** event to read them.

Inheritance Hierarchy

- **TMemDataSet**
- **TCustomDADataset**
- **TCustomPgDataSet**
- **TpgQuery**

See Also

- [Updating Data with PgDAC Dataset Components](#)
- [Master/Detail Relationships](#)
- [TpgStoredProc](#)
- [TpgTable](#)

© 1997-2021 Devart. All Rights Reserved.

5.14.1.20.1 Members

**TpgQuery** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BaseSQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>Cursor</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to fetch data from the REFCURSOR parameter or REFCURSOR field.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong></td>
<td>(inherited from <strong>TCustomPgDataset</strong>) Used to refresh record by RETURNING clause when insert or update is performed.</td>
</tr>
<tr>
<td><strong>FetchAll</strong></td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td><strong>FetchRows</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td><strong>FilterSQL</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from <strong>TMemDataset</strong>) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>IsQuery</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong></td>
<td>(inherited from <strong>TMemDataset</strong>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeyFields</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to build SQL</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KeySequence</td>
<td>Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.</td>
</tr>
<tr>
<td>LastInsertOID</td>
<td>Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>LockMode</td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of TCustomPgDataSet object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ParamCount</td>
<td>(inherited from TCustomDADataSet) Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td>(inherited from TCustomPgDataSet) Contains parameters for a query's SQL statement.</td>
</tr>
<tr>
<td>Prepared</td>
<td>(inherited from TMemDataSet) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>(inherited from TMemDataSet) Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>(inherited from TCustomDADataSet) Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>(inherited from TCustomDADataSet) Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>(inherited from TCustomDADataSet) Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SequenceMode</td>
<td>(inherited from TCustomPgDataSet) Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>SQL</td>
<td>(inherited from TCustomDADataSet) Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>(inherited from TCustomDADataSet) Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>(inherited from TCustomDADataSet) Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock</td>
<td>(inherited from TCustomDADataSet) Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>(inherited from TCustomDADataSet) Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
</tbody>
</table>
### PostgreSQL Data Access Components

**SQLRefresh** (inherited from **TCustomDADataset**)  
Used to specify a SQL statement that will be used to refresh current record by calling the `TCustomDADataset.RefreshRecord` procedure.

**SQLUpdate** (inherited from **TCustomDADataset**)  
Used to specify a SQL statement that will be used when applying an update to a dataset.

**UniDirectional** (inherited from **TCustomDADataset**)  
Used if an application does not need bidirectional access to records in the result set.

**UpdateObject** (inherited from **TCustomPgDataSet**)  
Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.

**UpdateRecordTypes** (inherited from **TMemDataSet**)  
Used to indicate the update status for the current record when cached updates are enabled.

**UpdatesPending** (inherited from **TMemDataSet**)  
Used to check the status of the cached updates buffer.

**UpdatingTable**  
Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddWhere</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td><strong>ApplyRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Writes dataset's pending cached</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BreakExec</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>CreateBlobStream</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td>CreateProcCall</td>
<td>Generates the stored procedure call.</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Fetched</td>
<td>Used to find out whether TCustomDADataSet has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>Used to learn whether TCustomDADataSet is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>Used to learn whether TCustomDADataSet is still fetching rows.</td>
</tr>
</tbody>
</table>
**FindKey** (inherited from **TCustomDADataSet**)  
TCustomDADataSet is fetching all rows to the end.  
Searches for a record which contains specified field values.

**FindMacro** (inherited from **TCustomDADataSet**)  
Finds a macro with the specified name.

**FindNearest** (inherited from **TCustomDADataSet**)  
Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.

**FindParam** (inherited from **TCustomPgDataSet**)  
Searches for and returns a parameter with the specified name.

**GetBlob** (inherited from **TMemDataSet**)  
Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

**GetDataType** (inherited from **TCustomDADataSet**)  
Returns internal field types defined in the MemData and accompanying modules.

**GetFieldObject** (inherited from **TCustomDADataSet**)  
Returns a multireference shared object from field.

**GetFieldPrecision** (inherited from **TCustomDADataSet**)  
Retrieves the precision of a number field.

**GetFieldScale** (inherited from **TCustomDADataSet**)  
Retrieves the scale of a number field.

**GetKeyFieldNames** (inherited from **TCustomDADataSet**)  
Provides a list of available key field names.

**GetOrderBy** (inherited from **TCustomDADataSet**)  
Retrieves an ORDER BY clause from a SQL statement.

**GetPgCursor** (inherited from **TCustomPgDataSet**)  
Retrieves a TPgCursor object for a field with known name.

**GetPgDate** (inherited from **TCustomPgDataSet**)  
Retrieves a TPgDate object for a field with known name.

**GetPgInterval** (inherited from **TCustomPgDataSet**)  
Retrieves a TPgInterval object for a field with known name.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetPgLargeObject</td>
<td>Retrieves a TPgLargeObject object for a field with known name. Inherited from TCustomPgDataSet.</td>
</tr>
<tr>
<td>GetPgRow</td>
<td>Retrieves a TPgRow object for a field with known name. Inherited from TCustomPgDataSet.</td>
</tr>
<tr>
<td>GetPgTime</td>
<td>Retrieves a TPgTime object for a field with known name. Inherited from TCustomPgDataSet.</td>
</tr>
<tr>
<td>GetPgTimeStamp</td>
<td>Retrieves a TPgTimeStamp object for a field with known name. Inherited from TCustomPgDataSet.</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>Sets the current record in this dataset similar to the current record in another dataset. Inherited from TCustomDADataset.</td>
</tr>
<tr>
<td>Locate</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it. Inherited from TMemDataSet.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet. Inherited from TMemDataSet.</td>
</tr>
<tr>
<td>Lock</td>
<td>Locks the current record. Inherited from TCustomDADataset.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name. Inherited from TCustomDADataset.</td>
</tr>
<tr>
<td>OpenNext</td>
<td>Opens the next REFCURSOR for stored procedure that returns more than one cursor. Inherited from TCustomPgDataSet.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Searches for and returns a parameter with the specified name. Inherited from TCustomPgDataSet.</td>
</tr>
<tr>
<td>Prepare</td>
<td>Allocates, opens, and parses cursor for a query. Inherited from TCustomDADataset.</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>Actualizes field values for the current record. Inherited from TCustomDADataset.</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy. Inherited from TCustomDADataset.</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>Marks all records in the cache of updates as unapplied. Inherited from TMemDataSet.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SaveSQL</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>TMemDataset</td>
</tr>
<tr>
<td>SetOrderBy</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>SetRange</td>
<td>TMemDataset</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>TMemDataset</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>TMemDataset</td>
</tr>
<tr>
<td>SQLSaved</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>UnLock</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>TMemDataset</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>TMemDataset</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>TMemDataset</td>
</tr>
</tbody>
</table>
Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong> (inherited from TCustomDADataSet)</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td><strong>AfterFetch</strong> (inherited from TCustomDADataSet)</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td><strong>AfterUpdateExecute</strong> (inherited from TCustomDADataSet)</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td><strong>BeforeFetch</strong> (inherited from TCustomDADataSet)</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td><strong>BeforeUpdateExecute</strong> (inherited from TCustomDADataSet)</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td><strong>OnUpdateError</strong> (inherited from TMemDataSet)</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td><strong>OnUpdateRecord</strong> (inherited from TMemDataSet)</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

Properties of the TPgQuery class.

For a complete list of the TPgQuery class members, see the TPgQuery Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BaseSQL</strong> (inherited from TCustomDADataSet)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong> (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Conditions</td>
<td>TCustomADataset</td>
</tr>
<tr>
<td>Connection</td>
<td>TCustomADataset</td>
</tr>
<tr>
<td>Cursor</td>
<td>TCustomPgDataset</td>
</tr>
<tr>
<td>DataTypeMap</td>
<td>TCustomADataset</td>
</tr>
<tr>
<td>Debug</td>
<td>TCustomADataset</td>
</tr>
<tr>
<td>DetailFields</td>
<td>TCustomADataset</td>
</tr>
<tr>
<td>Disconnected</td>
<td>TCustomADataset</td>
</tr>
<tr>
<td>DMLRefresh</td>
<td>TCustomPgDataset</td>
</tr>
<tr>
<td>FetchRows</td>
<td>TCustomADataset</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>TCustomADataset</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>TCustomADataset</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>TMemDataset</td>
</tr>
</tbody>
</table>
| IsQuery             | TCustomADataset         | Used to check whether SQL

© 2021 Devart
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KeyExclusive</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeyFields</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td><strong>KeySequence</strong></td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.</td>
</tr>
<tr>
<td><strong>LastInsertOID</strong></td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MacroCount</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>Macros</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>MasterSource</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Used to specify the behaviour of TCustomPgDataSet object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify whether parameters for the <strong>Params</strong> property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to indicate how many parameters are there in the <strong>Params</strong> property.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Contains parameters for a query's SQL statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SequenceMode</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td><strong>SQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td><strong>SQLInsert</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td><strong>SQLLock</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td><strong>SQLRecCount</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify the SQL statement that is used to get</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FetchAll</td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td>LockMode</td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
</tr>
<tr>
<td>UpdatingTable</td>
<td>Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete</td>
</tr>
</tbody>
</table>
5.14.1.20.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

TPgQuery

Syntax

property FetchAll: boolean;

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

Opening a dataset in FetchAll = False mode requires an active transaction. When the FetchAll property is False, the first call to TMemDataSet.Locate and TMemDataSet.LocateEx methods may take a lot of time to retrieve additional records to the client side.

5.14.1.20.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

TPgQuery
Syntax

```property`` LockMode: `TLockMode`;

Remarks

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is `lmNone`.

See Also

- `TPgStoredProc.LockMode`
- `TPgTable.LockMode`

© 1997-2021 Devart. All Rights Reserved.

5.14.1.20.2.3 UpdatingTable Property

Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

Class

`TPgQuery`

Syntax

```property`` UpdatingTable: `string`;

Remarks

Use the UpdatingTable property to specify which table in a query is assumed to be the target for the subsequent data-modification queries as a result of user incentive to insert, update or delete records.

This property is used on Insert, Update, Delete or RefreshRecord (see also `TCustomPgDataSet.Options`) if appropriate SQL (SQLInsert, SQLUpdate or SQLDelete) is not provided.
If UpdatingTable is not set then the first table used in a query is assumed to be the target.

Example

Below are two examples for the query, where:
1. the only allowed value for UpdatingTable property is 'Dept';
2. allowed values for UpdatingTable are 'Dept' and 'Emp'.

In the first case (or by default) editable field is ShipName, in the second - all fields from Emp.

1) Example 1.  
   SELECT * FROM Dept
2) Example 2.  
   SELECT * FROM Dept, Emp
   WHERE Dept.DeptNo = Emp.DeptNo

5.14.1.21 TPgSQL Class

A component for executing SQL statements and calling stored procedures on the database server.

For a list of all members of this type, see TPgSQL members.

Unit

PgAccess

Syntax

TPgSQL = class(TCustomDASQL);

Remarks

The TPgSQL component is a direct descendant of the TCustomDASQL class.

Use The TPgSQL component when a client application must execute SQL statement or the PL/SQL block, and call stored procedure on the database server. The SQL statement should not retrieve rows from the database.

Inheritance Hierarchy

TCustomDASQL
   TPgSQL
See Also
• TPgQuery

TPgSQL class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>Enables or disables changing screen cursor when executing commands in the NonBlocking mode.</td>
</tr>
<tr>
<td>CommandTimeout</td>
<td>The time to wait for a statement to be executed.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object that will be used to connect to a data store.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return a SQL statement with expanded macros.</td>
</tr>
<tr>
<td>LastInsertOID</td>
<td>Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.</td>
</tr>
</tbody>
</table>
### PostgreSQL Data Access Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ParamCount</strong></td>
<td>Indicates the number of parameters in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Contains parameters for a query's SQL statement.</td>
</tr>
<tr>
<td><strong>ParamValues</strong></td>
<td>Used to get or set the values of individual field parameters that are identified by name.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>Used to indicate whether a query is prepared for execution.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.</td>
</tr>
<tr>
<td><strong>UnpreparedExecute</strong></td>
<td>Used to apply simple executing to a SQL statement.</td>
</tr>
<tr>
<td><strong>UseParamTypes</strong></td>
<td>Used to disable automatic detection of the parameters data types.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BreakExec</strong></td>
<td>Breaks execution of an SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td>Checks whether TCustomDASQL still executes a SQL statement.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>Searches for and returns a parameter with the specified</td>
</tr>
</tbody>
</table>
### Properties of the TPgSQL class

For a complete list of the TPgSQL class members, see the TPgSQL Members topic.

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeCursor</td>
<td>Enables or disables changing screen cursor when executing commands in the NonBlocking mode.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return a SQL statement.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LastInsertOID</td>
<td>Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>Indicates the number of parameters in the Params property.</td>
</tr>
<tr>
<td>ParamValues</td>
<td>Used to get or set the values of individual field parameters that are identified by name.</td>
</tr>
<tr>
<td>Prepared</td>
<td>Used to indicate whether a query is prepared for execution.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommandTimeout</td>
<td>The time to wait for a statement to be executed.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object that will be used to connect to a data</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Contains parameters for a query's SQL statement.</td>
</tr>
<tr>
<td><strong>UnpreparedExecute</strong></td>
<td>Used to apply simple executing to a SQL statement.</td>
</tr>
<tr>
<td><strong>UseParamTypes</strong></td>
<td>Used to disable automatic detection of the parameters data types.</td>
</tr>
</tbody>
</table>

**See Also**
- [TPgSQL Class](#)
- [TPgSQL Class Members](#)

5.14.1.21.2.1 CommandTimeout Property

The time to wait for a statement to be executed.

**Class**

**TPgSQL**

**Syntax**

```property CommandTimeout: integer default 0;```

5.14.1.21.2.2 Connection Property

Used to specify a connection object that will be used to connect to a data store.

**Class**

**TPgSQL**

**Syntax**

```property Connection: TPgConnection;```
Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TPgConnection objects.

At run-time, set the Connection property to reference an existing TPgConnection object.

See Also

• TPgConnection

5.14.1.21.2.3 LastInsertOID Property

Returns OID of the record inserted by the last query for table with OIDs.

Class

TPgSQL

Syntax

property LastInsertOID: Int64;

Remarks

Use the LastInsertOID property to get OID of the record inserted by the last query for table with OIDs.

5.14.1.21.2.4 Params Property

Contains parameters for a query’s SQL statement.

Class

TPgSQL

Syntax
property Params: TPgParams stored False;

Remarks
Contains parameters for a query’s SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information). Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

5.14.1.21.2.5 UnpreparedExecute Property

Used to apply simple executing to a SQL statement.

Class
TPgSQL

Syntax
property UnpreparedExecute: boolean default False;

Remarks
If the UnpreparedExecute property is set to True, the simple execute is used for SQL statement. Statement is not prepared before execute. It allows to add multiple statements separated by semicolon to the SQL property.

5.14.1.21.2.6 UseParamTypes Property

Used to disable automatic detection of the parameters data types.

Class
TPgSQL
Syntax

```
property UseParamTypes: boolean default False;
```

Remarks

Set the UseParamTypes option to True to disable automatic detection of parameter types. When this option is True, data types of parameters are set basing on the DataType property. When this option is False, data types of the parameters are detected by server automatically.

5.14.1.21.3 Methods

Methods of the TPGSQL class.

For a complete list of the TPGSQL class members, see the TPGSQL Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BreakExec (inherited from TCustomDASQL)</td>
<td>Breaks execution of an SQL statement on the server.</td>
</tr>
<tr>
<td>Execute (inherited from TCustomDASQL)</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing (inherited from TCustomDASQL)</td>
<td>Checks whether TCustomDASQL still executes a SQL statement.</td>
</tr>
<tr>
<td>FindMacro (inherited from TCustomDASQL)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindParam</td>
<td>Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td>MacroByName (inherited from TCustomDASQL)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td>Prepare (inherited from TCustomDASQL)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>UnPrepare (inherited from TCustomDASQL)</td>
<td>Frees the resources allocated for a previously</td>
</tr>
</tbody>
</table>
See Also

- TPgSQL Class
- TPgSQL Class Members

© 1997-2021
Devart. All Rights Reserved.

5.14.1.21.3.1  FindParam Method

Searches for and returns a parameter with the specified name.

**Class**

TPgSQL

**Syntax**

```plaintext
function FindParam(const Value: string): TPgParam;
```

**Parameters**

- **Value**
  
  Holds the stored procedure name.

**Return Value**

- the parameter, if a match was found. Nil otherwise.

**Remarks**

Call the FindParam method to find a parameter with the name passed in the Name argument.
If a match was found, FindParam returns the parameter. Otherwise, it returns nil.

**See Also**

- TPgParam
- ParamByName

© 1997-2021
Devart. All Rights Reserved.
5.14.1.21.3.2  ParamByName Method

Searches for and returns a parameter with the specified name.

Class

TPgSQL

Syntax

```delphi
function ParamByName(const Value: string): TPgParam;
```

Parameters

Value

Holds the parameter name.

Return Value

the parameter, if a match was found. Otherwise an exception is raised.

Remarks

Call the ParamByName method to find a parameter with the name passed in the Name argument.

If a match is found, ParamByName returns the parameter. Otherwise, an exception is raised.

See Also

- TPgParam
- FindParam

© 1997-2021
Devart. All Rights Reserved.

5.14.1.22  TPgStoredProc Class

A component for accessing and executing stored procedures and functions.

For a list of all members of this type, see TPgStoredProc members.

Unit

PgAccess

Syntax

```delphi
TPgStoredProc = class(TCustomPgStoredProc);
```
Remarks

Use TPgStoredProc to access stored procedures on the database server.

You need only to define the StoredProcName property, and the SQL statement to call the stored procedure will be generated automatically.

Use the Execute method at runtime to generate request that instructs server to execute procedure and PrepareSQL to describe parameters at run time.

Inheritance Hierarchy

- TMemDataSet
  - TCustomDADataSet
    - TCustomPgDataSet
      - TCustomPgStoredProc
    - TPgStoredProc

See Also
- TPgQuery
- TPgSQL
- Updating Data with PgDAC Dataset Components

© 1997-2021 Devart. All Rights Reserved.

5.14.1.22.1 Members

**TPgStoredProc** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL (inherited from TCustomDADataSet)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td>CachedUpdates (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>CommandTimeout</td>
<td>The time to wait for a statement to be executed.</td>
</tr>
<tr>
<td><strong>Conditions</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>Cursor</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to fetch data from the REFCURSOR parameter or REFCURSOR field.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to refresh record by RETURNING clause when insert or update is performed.</td>
</tr>
<tr>
<td><strong>FetchAll</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td><strong>FetchRows</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td><strong>FilterSQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to get or set the list of</td>
</tr>
<tr>
<td>Field Name</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>IsQuery</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>KeyFields</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>KeySequence</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>LastInsertOID</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LockMode</td>
<td></td>
</tr>
<tr>
<td>MacroCount</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Macros</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>MasterFields</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>MasterSource</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to specify the behaviour of TCustomPgDataSet object.</td>
</tr>
<tr>
<td><strong>Overload</strong> (inherited from <strong>TCustomPgStoredProc</strong>)</td>
<td>Used to specify the overloading number.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Contains parameters for a query’s SQL statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td><strong>SequenceMode</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td><strong>SQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td><strong>SQLDelete</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQLInsert (inherited from TCustomDADataset)</td>
<td>Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount (inherited from TCustomDADataset)</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLUpdate (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>StoredProcName</td>
<td>Used to specify the name of the stored procedure to call on the server.</td>
</tr>
<tr>
<td>UniDirectional (inherited from TCustomDADataset)</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject (inherited from TCustomPgDataSet)</td>
<td>Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.</td>
</tr>
<tr>
<td>UpdateRecordTypes (inherited from TMemDataSet)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending (inherited from TMemDataSet)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>AddWhere</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>ApplyRange</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>BreakExec</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>CancelRange</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>CreateBlobStream</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>CreateProcCall</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>DeleteWhere</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>ExecProc</td>
<td>TCustomPgStoredProc</td>
</tr>
<tr>
<td>Execute</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Executing</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Fetched</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Fetching</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FindKey</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FindMacro</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FindNearest</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>FindParam</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GetBlob</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>GetDataType</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>GetPgCursor</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GetPgDate</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GetPgInterval</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GetPgLargeObject</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GetPgRow</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GetPgTime</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GetPgTimeStamp</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Locate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LocateEx</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Lock</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>MacroByName</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>OpenNext</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>ParamByName</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Prepare</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>PrepareSQL</td>
<td>TCustomPgStoredProc</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SaveSQL</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SetOrderBy</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SetRange</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SQLSaved</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>UnLock</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>(inherited from TMemDataSet) Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td>UpdateStatus</td>
<td>(inherited from TMemDataSet) Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>(inherited from TCustomDADataset) Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>AfterFetch</td>
<td>(inherited from TCustomDADataset) Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td>AfterUpdateExecute</td>
<td>(inherited from TCustomDADataset) Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td>BeforeFetch</td>
<td>(inherited from TCustomDADataset) Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td>BeforeUpdateExecute</td>
<td>(inherited from TCustomDADataset) Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td>OnUpdateError</td>
<td>(inherited from TMemDataSet) Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord</td>
<td>(inherited from TMemDataSet) Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>
Properties of the **TPgStoredProc** class.

For a complete list of the **TPgStoredProc** class members, see the [TPgStoredProc Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BaseSQL</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from <strong>TMemDataset</strong>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>Cursor</strong></td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Used to fetch data from the REFCURSOR parameter or REFCURSOR field.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong></td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Used to refresh record by RETURNING clause when insert or update is performed.</td>
</tr>
<tr>
<td><strong>FetchAll</strong></td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Defines whether to request all records of the query from</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FetchRows</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>IsQuery</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>KeySequence</td>
<td>Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.</td>
</tr>
<tr>
<td>LastInsertOID</td>
<td>Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>Field</td>
<td>Inherited From</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>MacroCount</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Macros</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>MasterFields</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>MasterSource</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Options</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>Overload</td>
<td>TCustomPgStoredProc</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>ParamCount</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Params</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>Prepared</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Ranged</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SequenceMode</td>
<td>Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>Used to specify a SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh</td>
<td>Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject</td>
<td>Used to point to an update object component which provides SQL statements that perform updates of read-only datasets.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>Used to indicate the update status for the current record when cached updates are</td>
</tr>
</tbody>
</table>

© 2021 Devart
enabled.

**UpdatesPending** (inherited from **TMemDataSet**) Used to check the status of the cached updates buffer.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CommandTimeout</strong></td>
<td>The time to wait for a statement to be executed.</td>
</tr>
<tr>
<td><strong>LockMode</strong></td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
</tr>
<tr>
<td><strong>StoredProcName</strong></td>
<td>Used to specify the name of the stored procedure to call on the server.</td>
</tr>
</tbody>
</table>

### See Also
- **TPgStoredProc Class**
- **TPgStoredProc Class Members**

---

5.14.1.22.2.1  **CommandTimeout Property**

The time to wait for a statement to be executed.

### Class

**TPgStoredProc**

### Syntax

```pascal
property CommandTimeout: integer;
```

---

5.14.1.22.2.2  **LockMode Property**

Used to specify what kind of lock will be performed when editing a record.

### Class

© 2021 Devart
**TPgStoredProc**

**Syntax**

```plaintext
property LockMode: TLockMode;
```

**Remarks**

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is lmNone.

---

**TPgTable Class**

A component for retrieving and updating data in a single table without writing SQL statements.

For a list of all members of this type, see TPgTable members.
Unit

PgAccess

Syntax

TPgTable = class(TCustomPgTable);

Remarks

The TPgTable component allows retrieving and updating data in a single table without writing SQL statements. Use TPgTable to access data in a table. Use the TableName property to specify table name. TPgTable uses the KeyFields property to build SQL statements for updating table data. KeyFields is a string containing a semicolon-delimited list of the field names.

Inheritance Hierarchy

TMemDataSet
    TCustomDADataSet
        TCustomPgDataSet
            TCustomPgTable
                TPgTable

See Also

- Updating Data with PgDAC Dataset Components
- Master/Detail Relationships
- TCustomPgDataSet
- TPgQuery
- TCustomPgTable

© 1997-2021
Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback
<table>
<thead>
<tr>
<th>BaseSQL (inherited from TCustomDADataSet)</th>
<th>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>Conditions (inherited from TCustomDADataSet)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td>Connection (inherited from TCustomDADataSet)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td>Cursor (inherited from TCustomPgDataSet)</td>
<td>Used to fetch data from the REFCURSOR parameter or REFCURSOR field.</td>
</tr>
<tr>
<td>DataTypeMap (inherited from TCustomDADataSet)</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td>Debug (inherited from TCustomDADataSet)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>DetailFields (inherited from TCustomDADataSet)</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td>Disconnected (inherited from TCustomDADataSet)</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td>DMLRefresh (inherited from TCustomPgDataSet)</td>
<td>Used to refresh record by RETURNING clause when insert or update is performed.</td>
</tr>
<tr>
<td>FetchAll</td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td>FetchRows (inherited from TCustomDADataSet)</td>
<td>Used to define the number of rows to be transferred across the network at the same time.</td>
</tr>
<tr>
<td>FilterSQL (inherited from TCustomDADataSet)</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FinalSQL (inherited from TCustomDADataset)</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>IndexFieldNames (inherited from TMemDataSet)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>IsQuery (inherited from TCustomDADataset)</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive (inherited from TMemDataSet)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields (inherited from TCustomDADataset)</td>
<td>Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td>KeySequence (inherited from TCustomPgDataSet)</td>
<td>Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.</td>
</tr>
<tr>
<td>LastInsertOID (inherited from TCustomPgDataSet)</td>
<td>Returns OID of the record inserted by the last query for table with OIDs.</td>
</tr>
<tr>
<td>Limit (inherited from TCustomPgTable)</td>
<td>Used to set the number of rows retrieved from the query.</td>
</tr>
<tr>
<td>LocalConstraints (inherited from TMemDataSet)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate (inherited from TMemDataSet)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>LockMode</td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
</tr>
<tr>
<td>MacroCount (inherited from TCustomDADataset)</td>
<td>Used to get the number of macros associated with the query.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td>Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Offset</td>
<td>Used to allow retrieving data from the server starting from the specified row.</td>
</tr>
<tr>
<td>Options</td>
<td>Used to specify the behaviour of TCustomPgDataSet object.</td>
</tr>
<tr>
<td>OrderFields</td>
<td>Used to build ORDER BY clause of SQL statements.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td>ParamCount</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td>Contains parameters for a query's SQL statement.</td>
</tr>
<tr>
<td>Prepared</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>Used to indicate the number of rows which were inserted,</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SequenceMode</td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>SQL</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify a SQL statement that will be used to refresh current record by calling the <strong>TCustomDADataset.RefreshRecord</strong> procedure.</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>TableName</td>
<td>Used to specify the name of the database table this component encapsulates.</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
</tbody>
</table>
| UpdateObject        | (inherited from **TCustomPgDataSet**) Used to point to an update object component which provides SQL statements that perform updates of updated, or deleted during the last query operation.
### UpdateRecordTypes (inherited from TMemDataSet)
Used to indicate the update status for the current record when cached updates are enabled.

### UpdatesPending (inherited from TMemDataSet)
Used to check the status of the cached updates buffer.

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddWhere (inherited from TCustomDADataset)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td>ApplyRange (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>BreakExec (inherited from TCustomDADataset)</td>
<td>Breaks execution of the SQL statement on the server.</td>
</tr>
<tr>
<td>CancelRange (inherited from TMemDataSet)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates (inherited from TMemDataSet)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates (inherited from TMemDataSet)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>CreateBlobStream (inherited from TCustomDADataset)</td>
<td>Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.</td>
</tr>
<tr>
<td>CreateProcCall (inherited from TCustomPgDataSet)</td>
<td>Generates the stored procedure call.</td>
</tr>
<tr>
<td>DeferredPost (inherited from TMemDataSet)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere (inherited from TCustomDADataset)</td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Fetched</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td>Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td>Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.</td>
</tr>
<tr>
<td>FindParam</td>
<td>Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>GetDataType</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>Returns a multireference</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GetPgCursor</td>
<td>Retrieves a TPgCursor object for a field with known name.</td>
</tr>
<tr>
<td>GetPgDate</td>
<td>Retrieves a TPgDate object for a field with known name.</td>
</tr>
<tr>
<td>GetPgInterval</td>
<td>Retrieves a TPgInterval object for a field with known name.</td>
</tr>
<tr>
<td>GetPgLargeObject</td>
<td>Retrieves a TPgLargeObject object for a field with known name.</td>
</tr>
<tr>
<td>GetPgRow</td>
<td>Retrieves a TPgRow object for a field with known name.</td>
</tr>
<tr>
<td>GetPgTime</td>
<td>Retrieves a TPgTime object for a field with known name.</td>
</tr>
<tr>
<td>GetPgTimeStamp</td>
<td>Retrieves a TPgTimeStamp object for a field with known name.</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td>Locate</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Lock</td>
<td>Locks the current record.</td>
</tr>
</tbody>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Method</th>
<th>Inherited From</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MacroByName</td>
<td>TCustomDADataSet</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>OpenNext</td>
<td>TCustomPgDataSet</td>
<td>Opens the next REFCURSOR for stored procedure that returns more than one cursor.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>TCustomPgDataSet</td>
<td>Searches for and returns a parameter with the specified name.</td>
</tr>
<tr>
<td>Prepare</td>
<td>TCustomDADataSet</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>TCustomDADataSet</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>TCustomDADataSet</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>TMemDataSet</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>TMemDataSet</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>SaveSQL</td>
<td>TCustomDADataSet</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>TMemDataSet</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td>SetOrderBy</td>
<td>TCustomDADataSet</td>
<td>Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td>SetRange</td>
<td>TMemDataSet</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td>SetRangeEnd</td>
<td>TMemDataSet</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td>SetRangeStart</td>
<td>TMemDataSet</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
</tbody>
</table>
**SQLSaved** (inherited from **TCustomDADataSet**)  
Determines if the **SQL** property value was saved to the **BaseSQL** property.

**UnLock** (inherited from **TCustomDADataSet**)  
Releases a record lock.

**UnPrepare** (inherited from **TMemDataSet**)  
Frees the resources allocated for a previously prepared query on the server and client sides.

**UpdateResult** (inherited from **TMemDataSet**)  
Reads the status of the latest call to the **ApplyUpdates** method while cached updates are enabled.

**UpdateStatus** (inherited from **TMemDataSet**)  
Indicates the current update status for the dataset when cached updates are enabled.

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td><strong>AfterFetch</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td><strong>AfterUpdateExecute</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td><strong>BeforeFetch</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td><strong>BeforeUpdateExecute</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td><strong>OnUpdateError</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td><strong>OnUpdateRecord</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>
Properties of the **TPgTable** class.

For a complete list of the **TPgTable** class members, see the [TPgTable Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BaseSQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<tr>
<td><strong>CachedUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>Conditions</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>Connection</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify a connection object to use to connect to a data store.</td>
</tr>
<tr>
<td><strong>Cursor</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to fetch data from the REFCURSOR parameter or REFCURSOR field.</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td><strong>DetailFields</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.</td>
</tr>
<tr>
<td><strong>Disconnected</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>DMLRefresh</strong> (inherited from <strong>TCustomPgDataSet</strong>)</td>
<td>Used to refresh record by RETURNING clause when insert or update is</td>
</tr>
<tr>
<td>Method</td>
<td>Class</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>FetchRows</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FilterSQL</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>IsQuery</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>KeyFields</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>KeySequence</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>LastInsertOID</td>
<td>TCustomPgDataSet</td>
</tr>
<tr>
<td>Limit</td>
<td>TCustomPgTable</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>MacroCount</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><strong>Macros</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.</td>
</tr>
<tr>
<td><strong>MasterSource</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Offset</strong></td>
<td>(inherited from <strong>TCustomPgTable</strong>) Used to allow retrieving data from the server starting from the specified row.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Used to specify the behaviour of TCustomPgDataSet object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td><strong>ParamCount</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>(inherited from <strong>TCustomPgDataSet</strong>) Contains parameters for a query's SQL statement.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>ReadOnly</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td><strong>RefreshOptions</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to indicate when the</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>(inherited from TCustomDADataset) Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SequenceMode</td>
<td>(inherited from TCustomPgDataSet) Used to specify the methods used internally to generate a sequenced field.</td>
</tr>
<tr>
<td>SQL</td>
<td>(inherited from TCustomDADataset) Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>(inherited from TCustomDADataset) Used to specify a SQL statement that will be used when applying a deletion to a record.</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>(inherited from TCustomDADataset) Used to specify the SQL statement that will be used when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLLock</td>
<td>(inherited from TCustomDADataset) Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>(inherited from TCustomDADataset) Used to specify the SQL statement that is used to get the record count when opening a dataset.</td>
</tr>
<tr>
<td>SQLRefresh</td>
<td>(inherited from TCustomDADataset) Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataset.RefreshRecord procedure.</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>(inherited from TCustomDADataset) Used to specify a SQL statement that will be used when applying an update to a dataset.</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>(inherited from TCustomDADataset) Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject</td>
<td>(inherited from TCustomPgDataSet) Used to point to an update object component which provides SQL statements that perform updates of</td>
</tr>
</tbody>
</table>
**UpdateRecordTypes** (inherited from [TMemDataSet](#))

Used to indicate the update status for the current record when cached updates are enabled.

**UpdatesPending** (inherited from [TMemDataSet](#))

Used to check the status of the cached updates buffer.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FetchAll</strong></td>
<td>Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td><strong>LockMode</strong></td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
</tr>
<tr>
<td><strong>OrderFields</strong></td>
<td>Used to build ORDER BY clause of SQL statements.</td>
</tr>
<tr>
<td><strong>TableName</strong></td>
<td>Used to specify the name of the database table this component encapsulates.</td>
</tr>
</tbody>
</table>

### See Also
- [TPgTable Class](#)
- [TPgTable Class Members](#)

5.14.1.23.2.1  FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

### Class

[TPgTable](#)

### Syntax

```
property FetchAll: boolean;
```
Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

Opening a dataset in FetchAll = False mode requires an active transaction. When the FetchAll property is False, the first call to TMemDataSet.Locate and TMemDataSet.LocateEx methods may take a lot of time to retrieve additional records to the client side.

Class

TPgTable

Syntax

```
property LockMode: TLockMode;
```

Remarks

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is lmNone.

See Also

- TPgStoredProc.LockMode
- TPgQuery.LockMode

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

5.14.1.23.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.
5.14.1.23.2.3 OrderFields Property

Used to build ORDER BY clause of SQL statements.

Class
TPgTable

Syntax

```plaintext
property OrderFields: string;
```

Remarks
TPgTable uses the OrderFields property to build ORDER BY clause of SQL statements. To set several field names to this property separate them with commas.

TPgTable is reopened when OrderFields is being changed.

See Also
- TPgTable

© 1997-2021 Devart. All Rights Reserved.

5.14.1.23.2.4 TableName Property

Used to specify the name of the database table this component encapsulates.

Class
TPgTable

Syntax

```plaintext
property TableName: string;
```

Remarks
Use the TableName property to specify the name of the database table this component encapsulates. If TCustomDADataset.Connection is assigned at design time, select a valid table name from the TableName drop-down list in Object Inspector.
5.14.1.24 TPgTimeField Class

A class providing access to the PostgreSQL time fields.

For a list of all members of this type, see TPgTimeField members.

Unit
PgAccess

Syntax

TPgTimeField = class(TCustomPgTimeStampField);

Remarks

The TPgTimeFields class provides access to the PostgreSQL time fields.

Inheritance Hierarchy

TCustomPgTimeStampField
   TPgTimeField

Members

TPgTimeField class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsPgTime</td>
<td>Used to provide access to a TPgTime object.</td>
</tr>
<tr>
<td>AsPgTimeStamp</td>
<td>(inherited from TCustomPgTimeStampField) Used to provide access to a TPgTimeStamp object.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback
Properties of the **TPgTimeField** class.

For a complete list of the **TPgTimeField** class members, see the **TPgTimeField Members** topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsPgTime</strong></td>
<td>Used to provide access to a TPgTime object.</td>
</tr>
<tr>
<td><strong>AsPgTimeStamp</strong></td>
<td>Used to provide access to a TPgTimeStamp object. (inherited from TCustomPgTimeStampField)</td>
</tr>
</tbody>
</table>

#### See Also
- **TPgTimeField Class**
- **TPgTimeField Class Members**

© 1997-2021 Devart. All Rights Reserved.

5.14.1.24.2.1 AsPgTime Property

Used to provide access to a TPgTime object.

### Class

**TPgTimeField**

### Syntax

```
property AsPgTime: TPgTime;
```

### Remarks

Use the AsPgTime property to provide access to a TPgTime object you can use for manipulations with the time value.
5.14.1.25 TPgTimeStampField Class

A class providing access to the PostgreSQL timestamp fields.

For a list of all members of this type, see TPgTimeStampField members.

Unit

PgAccess

Syntax

TPgTimeStampField = class(TCustomPgTimeStampField);

Remarks

TPgTimeStampField provides access to PostgreSQL timestamp fields.

Inheritance Hierarchy

TCustomPgTimeStampField
    TPgTimeStampField

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsPgTimeStamp</td>
<td>Used to provide access to a TPgTimeStam object.</td>
</tr>
</tbody>
</table>

For a complete list of the TPgTimeStampField class members, see the TPgTimeStampField Members topic.
5.14.1.25.2.1  AsPgTimeStamp Property

Used to provide access to a TPgTimeStamp object.

Class

TPgTimeStampField

Syntax

**property** AsPgTimeStamp: TPgTimeStamp;

Remarks

Use the AsTimeStamp property to get access to a TPgTimeStamp object which you can use for manipulations with the timestamp value.
TPgTransaction = class(TDATransaction);

Remarks
The TPgTransaction component is used to provide discrete transaction control over connection. This component is used internally by PgDAC. For user, it is recommended to use methods of TPgConnection instead (such as StartTransaction, Commit, Rollback) for transaction control.

Inheritance Hierarchy
TDATransaction
  TPgTransaction

See Also
• TCustomDAConnection.StartTransaction
• TCustomDAConnection.Commit
• TCustomDAConnection.Rollback

© 1997-2021 Devart. All Rights Reserved.

TPgTransaction class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active (inherited from TDATransaction)</td>
<td>Used to determine if the transaction is active.</td>
</tr>
<tr>
<td>DefaultCloseAction (inherited from TDATransaction)</td>
<td>Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.</td>
</tr>
<tr>
<td>IsolationLevel</td>
<td>Used to specify how the transactions containing database modifications are handled.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commit</strong> (inherited from <strong>TDATransaction</strong>)</td>
<td>Commits the current transaction.</td>
</tr>
<tr>
<td><strong>Rollback</strong> (inherited from <strong>TDATransaction</strong>)</td>
<td>Discards all modifications of data associated with the current transaction and ends the transaction.</td>
</tr>
<tr>
<td><strong>StartTransaction</strong> (inherited from <strong>TDATransaction</strong>)</td>
<td>Begins a new transaction.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnCommit</strong> (inherited from <strong>TDATransaction</strong>)</td>
<td>Occurs after the transaction has been successfully committed.</td>
</tr>
<tr>
<td><strong>OnCommitRetaining</strong> (inherited from <strong>TDATransaction</strong>)</td>
<td>Occurs after CommitRetaining has been executed.</td>
</tr>
<tr>
<td><strong>OnError</strong> (inherited from <strong>TDATransaction</strong>)</td>
<td>Used to process errors that occur during executing a transaction.</td>
</tr>
<tr>
<td><strong>OnRollback</strong> (inherited from <strong>TDATransaction</strong>)</td>
<td>Occurs after the transaction has been successfully rolled back.</td>
</tr>
<tr>
<td><strong>OnRollbackRetaining</strong> (inherited from <strong>TDATransaction</strong>)</td>
<td>Occurs after RollbackRetaining has been executed.</td>
</tr>
</tbody>
</table>

Properties of the **TPgTransaction** class.

For a complete list of the **TPgTransaction** class members, see the **TPgTransaction Members** topic.

Public
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Used to determine if the transaction is active.</td>
</tr>
<tr>
<td>DefaultCloseAction</td>
<td>Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.</td>
</tr>
<tr>
<td>IsolationLevel</td>
<td>Used to specify how the transactions containing database modifications are handled.</td>
</tr>
</tbody>
</table>

**See Also**
- `TPgTransaction Class`
- `TPgTransaction Class Members`

5.14.1.26.2.1 IsolationLevel Property

Used to specify how the transactions containing database modifications are handled.

**Class**

`TPgTransaction`

**Syntax**

```property
property IsolationLevel: TCRIsolationLevel;
```

**Remarks**

Use the IsolationLevel property to specify how the transactions containing database modifications are handled.
5.14.1.27 TPgUpdateSQL Class

Lets you tune update operations for the DataSet component.

For a list of all members of this type, see TPgUpdateSQL members.

Unit

PgAccess

Syntax

TPgUpdateSQL = class(TCustomDAUpdateSQL);

Inheritance Hierarchy

TCustomDAUpdateSQL
  TPgUpdateSQL

5.14.1.27.1 Members

TPgUpdateSQL class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSet</td>
<td>Used to hold a reference to the TCustomDaDataSet object that is being updated.</td>
</tr>
<tr>
<td>DeleteObject</td>
<td>Provides ability to perform advanced adjustment of the delete operations.</td>
</tr>
<tr>
<td>DeleteSQL</td>
<td>Used when deleting a record.</td>
</tr>
<tr>
<td>InsertObject</td>
<td>Provides ability to perform advanced adjustment of insert operations.</td>
</tr>
<tr>
<td>InsertSQL</td>
<td>Used when inserting a record.</td>
</tr>
<tr>
<td>LockObject</td>
<td>Provides ability to perform advanced adjustment of lock operations.</td>
</tr>
</tbody>
</table>
**LockSQL** (inherited from **TCustomDAUpdateSQL**) | Used to lock the current record.  
**ModifyObject** (inherited from **TCustomDAUpdateSQL**) | Provides ability to perform advanced adjustment of modify operations.  
**ModifySQL** (inherited from **TCustomDAUpdateSQL**) | Used when updating a record.  
**RefreshObject** (inherited from **TCustomDAUpdateSQL**) | Provides ability to perform advanced adjustment of refresh operations.  
**RefreshSQL** (inherited from **TCustomDAUpdateSQL**) | Used to specify an SQL statement that will be used for refreshing the current record by **TCustomDADataset.RefreshRecord** procedure.  
**SQL** (inherited from **TCustomDAUpdateSQL**) | Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apply</strong> (inherited from <strong>TCustomDAUpdateSQL</strong>)</td>
<td>Sets parameters for a SQL statement and executes it to update a record.</td>
</tr>
<tr>
<td><strong>ExecSQL</strong> (inherited from <strong>TCustomDAUpdateSQL</strong>)</td>
<td>Executes a SQL statement.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback

### 5.14.2 Types

Types in the **PgAccess** unit.

**Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPgNoticeEvent</strong></td>
<td>This type is used for the <strong>TPgConnection.OnNotice</strong> event.</td>
</tr>
</tbody>
</table>
5.14.2.1 TPgNoticeEvent Procedure Reference

This type is used for the **TPgConnection.OnNotice** event.

Unit

**PgAccess**

Syntax

```object
TPgNoticeEvent = procedure (Sender: TObject; Errors: TPgErrors) of object;
```

Parameters

- **Sender**
  An object that raised the event.

- **Errors**
  Holds a collection of TPgError objects containing notices.

5.14.2.2 TPgNotificationEvent Procedure Reference

This type is used for the **TPgConnection.OnNotification** event.

Unit

**PgAccess**

Syntax

```object
TPgNotificationEvent = procedure (Sender: TObject; const EventName: string; PID: integer; const EventMessage: string) of object;
```

Parameters

- **Sender**
An object that raised the event.

*EventName*
Holds the event name.

*PID*
Holds the process ID of a connection that has sent this notification.

*EventMessage*
Holds the event message.

Remarks
EventMessage parameter supported in PostgreSQL 9.0 or higher. For previous PostgreSQL versions EventMessage will be always empty.

5.14.3 Enumerations
Enumerations in the *PgAccess* unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>TPgIsolationLevel</em></td>
<td>Specifies the way the transactions containing database modifications are handled.</td>
</tr>
</tbody>
</table>

5.14.3.1 *TPgIsolationLevel* Enumeration
Specifications the way the transactions containing database modifications are handled.

Unit
*PgAccess*

Syntax
```
TPgIsolationLevel = (pilReadCommitted, pilSerializable);
```
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>pilReadCommitted</td>
<td>If the transaction contains DML that requires row locks held by another transaction, then the DML statement waits until the row locks are released. The default PostgreSQL behavior.</td>
</tr>
<tr>
<td>pilSerializable</td>
<td>Specifies serializable transaction isolation mode as defined in the SQL92 standard. If a serializable transaction contains data manipulation language (DML) that attempts to update any resource that may have been updated in a transaction uncommitted at the start of the serializable transaction, then the DML statement fails.</td>
</tr>
</tbody>
</table>

5.14.4 Constants

Constants in the **PgAccess** unit.

**Constants**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgDACVersion</td>
<td>The version of PostgreSQL Data Access Components.</td>
</tr>
</tbody>
</table>

5.14.4.1 PgDACVersion Constant

The version of PostgreSQL Data Access Components.

**Unit**

**PgAccess**

**Syntax**

```plaintext
PgDACVersion = '6.4.1';
```
5.15 **PgAlerter**

This unit contains the implementation of the TPgAlerter component.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgAlerter</td>
<td>A component that allows to register interest in and asynchronously handle event notifications posted by PostgreSQL server.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.15.1 **Classes**

Classes in the **PgAlerter** unit.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgAlerter</td>
<td>A component that allows to register interest in and asynchronously handle event notifications posted by PostgreSQL server.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.15.1.1 **TPgAlerter Class**

A component that allows to register interest in and asynchronously handle event notifications posted by PostgreSQL server.

For a list of all members of this type, see **TPgAlerter** members.

**Unit**

**PgAlerter**

**Syntax**
TPgAlerter = class(TDAAlerte);
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendEvent</td>
<td>Overloaded. Description is not available at the moment.</td>
</tr>
<tr>
<td>Start</td>
<td>(inherited from TDAAlerter) Starts waiting process.</td>
</tr>
<tr>
<td>Stop</td>
<td>(inherited from TDAAlerter) Stops waiting process.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnError</td>
<td>(inherited from TDAAlerter) Occurs if an exception occurs in waiting process</td>
</tr>
<tr>
<td>OnEvent</td>
<td>Occurs when waiting thread receives event notification from PostgreSQL server.</td>
</tr>
</tbody>
</table>

Properties of the TPgAlerter class.

For a complete list of the TPgAlerter class members, see the TPgAlerter Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>(inherited from TDAAlerter) Used to determine if TDAAlerter waits for messages.</td>
</tr>
<tr>
<td>AutoRegister</td>
<td>(inherited from TDAAlerter) Used to automatically register events whenever connection opens.</td>
</tr>
<tr>
<td>Connection</td>
<td>(inherited from TDAAlerter) Used to specify the connection for TDAAlerter.</td>
</tr>
</tbody>
</table>
Events

Used to set the names of the events to wait.

See Also
- **TPgAlerter Class**
- **TPgAlerter Class Members**

Class

TPgAlerter

Syntax

```
property Events: string;
```

Remarks

Use the Events property to set the names of the events to wait.

Methods

Methods of the **TPgAlerter** class.

For a complete list of the **TPgAlerter** class members, see the **TPgAlerter Members** topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendEvent</td>
<td>Overloaded. Description is not available at the moment.</td>
</tr>
<tr>
<td><strong>Start</strong></td>
<td>(inherited from <strong>TDAAlerter</strong>) Starts waiting process.</td>
</tr>
<tr>
<td><strong>Stop</strong></td>
<td>(inherited from <strong>TDAAlerter</strong>) Stops waiting process.</td>
</tr>
</tbody>
</table>
5.15.1.3.1 SendEvent Method

Class

**TPgAlerter**

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SendEvent(const Name: string)</code></td>
<td>Sends an event notification with Name.</td>
</tr>
<tr>
<td><code>SendEvent(const Name: string; Message: string)</code></td>
<td>Sends an event notification with Name.</td>
</tr>
</tbody>
</table>

Sends an event notification with Name.

Class

**TPgAlerter**

Syntax

```plaintext
procedure SendEvent(const Name: string); overload;
```

Parameters

- **Name**
  - Message name.

Remarks

Call the SendEvent method to send an event notification with Name.
Sends an event notification with Name.

**Class**

TPgAlerter

**Syntax**

```plaintext
procedure SendEvent(const Name: string; Message: string);
```

**Parameters**

- **Name**
  - Message name.

- **Message**
  - Message text.

**Remarks**

Note: required PostgreSQL 9.0 or higher.

5.15.1.1.4 Events

Events of the TPgAlerter class.

For a complete list of the TPgAlerter class members, see the [TPgAlerter Members](#) topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onError</td>
<td>Occurs if an exception occurs in waiting process</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onEvent</td>
<td>Occurs when waiting thread receives event notification from PostgreSQL server.</td>
</tr>
</tbody>
</table>

**See Also**
5.15.1.4.1 OnEvent Event

Occurs when waiting thread receives event notification from PostgreSQL server.

Class

TPgAlerter

Syntax

property OnEvent: TPgNotificationEvent;

Remarks

OnEvent occurs when waiting thread receives event notification from PostgreSQL server.

5.16 PgClasses

This unit contains the implementation of internal PgDAC classes and types.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgCursor</td>
<td>A class for working with PostgreSQL cursors.</td>
</tr>
</tbody>
</table>

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TProtocolVersion</td>
<td>Specifies protocol version to be used when several versions are available.</td>
</tr>
<tr>
<td>TSSLMode</td>
<td>This option determines whether or with what priority an SSL connection will be</td>
</tr>
</tbody>
</table>
5.16.1 Classes

Classes in the PgClasses unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgCursor</td>
<td>A class for working with PostgreSQL cursors.</td>
</tr>
</tbody>
</table>

5.16.1.1 TPgCursor Class

A class for working with PostgreSQL cursors.

For a list of all members of this type, see TPgCursor members.

Unit
PgClasses

Syntax

```
TPgCursor = class(TCRCursor);
```

Remarks

The TPgCursor class is used to work with PostgreSQL cursors. This is a PgDAC internal class, and application should not use it directly.

Inheritance Hierarchy

TSharedObject
  TCRCursor
  TPgCursor
5.16.1.1.1 Members

**TPgCursor** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>State</td>
<td>Used to set the cursor state.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.16.1.2 Properties

Properties of the **TPgCursor** class.

For a complete list of the **TPgCursor** class members, see the [TPgCursor Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>State</td>
<td>Used to set the cursor state.</td>
</tr>
</tbody>
</table>

See Also
5.16.1.1.2.1 State Property

Used to set the cursor state.

Class

TPgCursor

Syntax

property State: TCursorState;

See Also

• TCursorState

5.16.2 Enumerations

Enumerations in the PgClasses unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TProtocolVersion</td>
<td>Specifies protocol version to be used when several versions are available.</td>
</tr>
<tr>
<td>TSSLMode</td>
<td>This option determines whether or with what priority an SSL connection will be negotiated with the server.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devar. All Rights Reserved.
5.16.2.1 TProtocolVersion Enumeration

Specifies protocol version to be used when several versions are available.

Unit
PgClasses

Syntax
TProtocolVersion = (pv20, pv30, pvAuto);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pv20</td>
<td>Set ProtocolVersion to pv20 to work with PostgreSQL server version 7.3 or older that don't support protocol version 3.0.</td>
<td></td>
</tr>
<tr>
<td>pv30</td>
<td>Set ProtocolVersion to pv30 to enforce protocol version 3.0.</td>
<td></td>
</tr>
<tr>
<td>pvAuto</td>
<td>Set ProtocolVersion to pvAuto to automatically select between protocol versions depending on the specific query for the best possible performance.</td>
<td></td>
</tr>
</tbody>
</table>

Remarks

The default value is pvAuto.

© 1997-2021 Devart. All Rights Reserved. 
Request Support DAC Forum Provide Feedback

5.16.2.2 TSSLMode Enumeration

This option determines whether or with what priority an SSL connection will be negotiated with the server.

Unit
PgClasses

Syntax
TSSLMode = (smDisable, smRequire, smPrefer, smAllow, smVerifyCA, smVerifyFull);

Values
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>smAllow</td>
<td>Negotiates trying first a non-SSL connection, then if that fails, tries an SSL connection.</td>
</tr>
<tr>
<td>smDisable</td>
<td>Only an unencrypted SSL connection will be attempted.</td>
</tr>
<tr>
<td>smPrefer</td>
<td>Negotiates trying first an SSL connection, then if that fails, tries a regular non-SSL connection.</td>
</tr>
<tr>
<td>smRequire</td>
<td>Tries only an SSL connection.</td>
</tr>
<tr>
<td>smVerifyCA</td>
<td>Verifies server identity by validating the server certificate chain up to the root certificate installed on the client machine.</td>
</tr>
<tr>
<td>smVerifyFull</td>
<td>Verifies server identity by validating the server certificate chain up to the root certificate installed on the client machine and validates that the server hostname matches the server certificate.</td>
</tr>
</tbody>
</table>

Remarks
The default value is smDisable.

If PostgreSQL is compiled without SSL support, using option smRequire will cause an error, while options smAllow and smPrefer will be accepted, but PgDAC will not in fact attempt an SSL connection.

5.17 PgConnectionPool

This unit contains the TPgConnectionPoolManager class for managing connection pool.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgConnectionPoolManager</td>
<td>A class of methods that are used for managing PgDAC connection pool.</td>
</tr>
</tbody>
</table>

5.17.1 Classes

Classes in the PgConnectionPool unit.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPgConnectionPoolManager</strong></td>
<td>A class of methods that are used for managing PgDAC connection pool.</td>
</tr>
</tbody>
</table>

5.17.1.1 **TPgConnectionPoolManager Class**

A class of methods that are used for managing PgDAC connection pool.

For a list of all members of this type, see [TPgConnectionPoolManager members](#).

**Unit**

`PgConnectionPool`

**Syntax**

```
TPgConnectionPoolManager = class(TCRConnectionPoolManager);
```

**Remarks**

Use the TPgConnectionPoolManager methods to manage PgDAC connection pool.

**Inheritance Hierarchy**

TCRConnectionPoolManager

TPgConnectionPoolManager

**See Also**

- [Connection Pooling](#)

© 1997-2021 Devart. All Rights Reserved.  
[Request Support](#)  [DAC Forum](#)  [Provide Feedback](#)
5.18 **PgDacVcl**

This unit contains the visual constituent of PgDAC.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgConnectDialog</td>
<td>A component providing a dialog box for user to supply his login information.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.18.1 **Classes**

Classes in the ** PgDacVcl ** unit.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgConnectDialog</td>
<td>A component providing a dialog box for user to supply his login information.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.18.1.1 **TPgConnectDialog Class**

A component providing a dialog box for user to supply his login information.

For a list of all members of this type, see **TPgConnectDialog** members.

**Unit**

**PgDacVcl**

**Syntax**

```
TPgConnectDialog = class(TCustomConnectDialog);
```

**Remarks**

The TPgConnectDialog class provides a dialog box for user to supply his login information.
The TPgConnectDialog component is a direct descendant of the TCustomConnectDialog class. Use TPgConnectDialog to provide dialog box for a user to supply user name, password, server, port, and database name. You may want to customize appearance of the dialog box using properties of this class.

Inheritance Hierarchy

TCustomConnectDialog
    TPgConnectDialog

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelButton</td>
<td>Used to specify the label for the Cancel button.</td>
</tr>
<tr>
<td>Caption</td>
<td>Used to set the caption of dialog box.</td>
</tr>
<tr>
<td>ConnectButton</td>
<td>Used to specify the label for the Connect button.</td>
</tr>
<tr>
<td>Connection</td>
<td>Holds the TPgConnection component which uses TPgConnectDialog object.</td>
</tr>
<tr>
<td>DatabaseLabel</td>
<td>Used to specify a prompt for database edit.</td>
</tr>
<tr>
<td>DialogClass</td>
<td>Used to specify the class of the form that will be displayed to enter login information.</td>
</tr>
<tr>
<td>LabelSet</td>
<td>Used to set the language of buttons and labels captions.</td>
</tr>
<tr>
<td>PasswordLabel</td>
<td>Used to specify a prompt for password edit.</td>
</tr>
<tr>
<td>PortLabel</td>
<td>Used to specify a prompt for port edit.</td>
</tr>
<tr>
<td>Retries</td>
<td>Used to indicate the number of retries of failed</td>
</tr>
</tbody>
</table>
### Properties

**SavePassword** (inherited from **TCustomConnectDialog**)  
Used for the password to be displayed in ConnectDialog in asterisks.

**ServerLabel** (inherited from **TCustomConnectDialog**)  
Used to specify a prompt for the server name edit.

**ShowDatabase**  
Used to display a field for entering database at connect dialog.

**ShowPort**  
Used to display a field for entering port at connect dialog.

**StoreLogInfo** (inherited from **TCustomConnectDialog**)  
Used to specify whether the login information should be kept in system registry after a connection was established.

**UsernameLabel** (inherited from **TCustomConnectDialog**)  
Used to specify a prompt for username edit.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execute</strong></td>
<td>Displays the connect dialog and calls the connection’s Connect method when user clicks the Connect button.</td>
</tr>
<tr>
<td><strong>GetServerList</strong></td>
<td>Retrieves a list of available server names.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CancelButton</td>
<td>Used to specify the label for the Cancel button.</td>
</tr>
<tr>
<td>Caption</td>
<td>Used to set the caption of dialog box.</td>
</tr>
<tr>
<td>ConnectButton</td>
<td>Used to specify the label for the Connect button.</td>
</tr>
<tr>
<td>Connection</td>
<td>Holds the TPgConnection component which uses TPgConnectDialog object.</td>
</tr>
<tr>
<td>DatabaseLabel</td>
<td>Used to specify a prompt for database edit.</td>
</tr>
<tr>
<td>DialogClass</td>
<td>Used to specify the class of the form that will be displayed to enter login information.</td>
</tr>
<tr>
<td>LabelSet</td>
<td>Used to set the language of buttons and labels captions.</td>
</tr>
<tr>
<td>PasswordLabel</td>
<td>Used to specify a prompt for password edit.</td>
</tr>
<tr>
<td>PortLabel</td>
<td>Used to specify a prompt for port edit.</td>
</tr>
<tr>
<td>Retries</td>
<td>Used to indicate the number of retries of failed connections.</td>
</tr>
<tr>
<td>SavePassword</td>
<td>Used for the password to be displayed in ConnectDialog in asterisks.</td>
</tr>
<tr>
<td>ServerLabel</td>
<td>Used to specify a prompt for the server name edit.</td>
</tr>
<tr>
<td>ShowDatabase</td>
<td>Used to display a field for entering database at connect dialog.</td>
</tr>
<tr>
<td>ShowPort</td>
<td>Used to display a field for entering port at connect dialog.</td>
</tr>
<tr>
<td>StoreLogInfo</td>
<td>Used to specify whether the login information should be kept in system registry after a connection was established.</td>
</tr>
<tr>
<td>UsernameLabel</td>
<td>Used to specify a prompt for username edit.</td>
</tr>
</tbody>
</table>
**TCustomConnectDialog**

See Also
- **TPgConnectDialog Class**
- **TPgConnectDialog Class Members**

5.18.1.1.2.1 Connection Property

Holds the TPgConnection component which uses TPgConnectDialog object.

Class

**TPgConnectDialog**

Syntax

```property
property Connection: TPgConnection;
```

Remarks

Use the Connection property to learn which TPgConnection component uses TPgConnectDialog object. This property is read-only.

See Also
- **TCustomDACConnection.ConnectDialog**

5.18.1.1.2.2 DatabaseLabel Property

Used to specify a prompt for database edit.

Class

**TPgConnectDialog**

Syntax

```property
property DatabaseLabel: string;
```
5.18.1.2.3  PortLabel Property

Used to specify a prompt for port edit.

Class

TPgConnectDialog

Syntax

```delphi
property PortLabel: string;
```

Remarks

Use the PortLabel property to specify a prompt for port edit.

See Also

- `ShowPort`

5.18.1.2.4  ShowDatabase Property

Used to display a field for entering database at connect dialog.

Class

TPgConnectDialog

Syntax

```delphi
property ShowDatabase: boolean default True;
```

Remarks

Use the DatabaseLabel property to specify a prompt for database edit.

See Also

- `ShowDatabase`
Use the ShowDatabase property to display a field for entering database at connect dialog.

The default value is True.

See Also
- DatabaseLabel
- ShowPort

5.19 PgDump

This unit contains the implementation of the TPgDump component.

Classes
TPgDump

A component for storing database or its parts as a script and also for restoring database from the received script.

TPgDumpOptions

This class allows setting up the behaviour of the TPgDump component.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgDumpObjects</td>
<td>Represents the set of TPgDumpObject.</td>
</tr>
</tbody>
</table>

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgDumpMode</td>
<td>Specifies the mode of backup performed by TPgDump.</td>
</tr>
<tr>
<td>TPgDumpObject</td>
<td>Specifies the types of objects that are backuped by PgDump.</td>
</tr>
</tbody>
</table>

5.19.1 Classes

Classes in the PgDump unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgDump</td>
<td>A component for storing database or its parts as a script and also for restoring database from the received script.</td>
</tr>
<tr>
<td>TPgDumpOptions</td>
<td>This class allows setting up the behaviour of the TPgDump component.</td>
</tr>
</tbody>
</table>
5.19.1.1 TPgDump Class

A component for storing database or its parts as a script and also for restoring database from the received script.

For a list of all members of this type, see TPgDump members.

Unit

PgDump

Syntax

TPgDump = class(TDADump);

Remarks

The TPgDump component is a component for storing database or its parts as a script and also for restoring database from the received script.

Serves to store a database or its parts as a script and also to restore database from received script. TPgDump behaviour is similar to pg_dump program. Use SchemaNames, TableNames, and ObjectTypes properties to specify the objects to be stored. To generate a script call TDADump.Backup or TDADump.BackupQuery method. Resulted script can be viewed in TDADump.SQL.

Inheritance Hierarchy

TDADump
TPgDump

5.19.1.1.1 Members

TPgDump class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 1997-2021 Devart. All Rights Reserved.
Connection (inherited from TDADump) | Used to specify a connection object that will be used to connect to a data store.
---|---
Debug (inherited from TDADump) | Used to display the statement that is being executed and the values and types of its parameters.
Mode | Used to specify the mode of backup performed by TPgDump.
ObjectTypes | Used to specify the types of objects that are backuped by PgDump.
Options | Specifies the behaviour of the TPqDump component.
SchemaNames | Used to set the names of the schemas to dump.
SQL (inherited from TDADump) | Used to set or get the dump script.
TableNames (inherited from TDADump) | Used to set the names of the tables to dump.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup (inherited from TDADump)</td>
<td>Dumps database objects to the TDADump.SQL property.</td>
</tr>
<tr>
<td>BackupQuery (inherited from TDADump)</td>
<td>Dumps the results of a particular query.</td>
</tr>
<tr>
<td>BackupToFile (inherited from TDADump)</td>
<td>Dumps database objects to the specified file.</td>
</tr>
<tr>
<td>BackupToStream (inherited from TDADump)</td>
<td>Dumps database objects to the stream.</td>
</tr>
<tr>
<td>Restore (inherited from TDADump)</td>
<td>Executes a script contained in the SQL property.</td>
</tr>
<tr>
<td>RestoreFromFile (inherited from TDADump)</td>
<td>Executes a script from a file.</td>
</tr>
<tr>
<td>RestoreFromStream (inherited from TDADump)</td>
<td>Executes a script received from the stream.</td>
</tr>
</tbody>
</table>

Events
### Properties

Properties of the **TPgDump** class.

For a complete list of the **TPgDump** class members, see the [TPgDump Members](#) topic.

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection</strong> (inherited from <strong>TDADump</strong>)</td>
<td>Used to specify a connection object that will be used to connect to a data store.</td>
</tr>
</tbody>
</table>

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Debug</strong> (inherited from <strong>TDADump</strong>)</td>
<td>Used to display the statement that is being executed and the values and</td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td>Used to specify the mode of backup performed by TPgDump.</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ObjectTypes</strong></td>
<td>Used to specify the types of objects that are backed up by PgDump.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Specifies the behaviour of the TPgDump component.</td>
</tr>
<tr>
<td><strong>SchemaNames</strong></td>
<td>Used to set the names of the schemas to dump.</td>
</tr>
<tr>
<td><strong>SQL</strong> (inherited from <strong>TDADump</strong>)</td>
<td>Used to set or get the dump script.</td>
</tr>
<tr>
<td><strong>TableNames</strong> (inherited from <strong>TDADump</strong>)</td>
<td>Used to set the names of the tables to dump.</td>
</tr>
</tbody>
</table>

**See Also**
- **TPgDump Class**
- **TPgDump Class Members**

5.19.1.2.1 Mode Property

Used to specify the mode of backup performed by TPgDump.

**Class**

**TPgDump**

**Syntax**

```property Mode: TPgDumpMode default dmAll;```

**Remarks**

Use the Mode property to specify the mode of backup performed by TPgDump.
5.19.1.1.2.2 ObjectTypes Property

Used to specify the types of objects that are backed up by PgDump.

Class

**TPgDump**

Syntax

```plaintext
property ObjectTypes: TPgDumpObjects default [doSchemas, doLanguages, doDomains, doTypes, doTables, doViews, doSequences, doStoredProcs, doTriggers, doIndexes];
```

Remarks

Use the ObjectTypes property to specify the types of objects that are backed up by PgDump.

5.19.1.1.2.3 Options Property

Specifies the behaviour of the TPgDump component.

Class

**TPgDump**

Syntax

```plaintext
property Options: TPgDumpOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TPgDump component.

Descriptions of all options are in the table below.

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateConstraints</td>
<td>Used to add statements to create table constraints to the dump.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback

© 2021 Devart
5.19.1.2.4 SchemaNames Property

Used to set the names of the schemas to dump.

Class
TPgDump

Syntax

property SchemaNames: string;

Remarks
Use the SchemaNames property to set the names of the schemas to dump. Table names must be separated with semicolons. If the property is empty, the Backup method will dump all available schemas.

5.19.1.2 TPgDumpOptions Class

This class allows setting up the behaviour of the TPgDump component.

For a list of all members of this type, see TPgDumpOptions members.

Unit
PgDump

Syntax

TPgDumpOptions = class(TDADumpOptions);
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddDrop (inherited from TDADumpOptions)</td>
<td>Used to add drop statements to a script before creating statements.</td>
</tr>
<tr>
<td>CompleteInsert (inherited from TDADumpOptions)</td>
<td>Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.</td>
</tr>
<tr>
<td>CreateConstraints</td>
<td>Used to add statements to create table constrains to the dump.</td>
</tr>
<tr>
<td>GenerateHeader (inherited from TDADumpOptions)</td>
<td>Used to add a comment header to a script.</td>
</tr>
<tr>
<td>QuoteNames (inherited from TDADumpOptions)</td>
<td>Used for TDADump to quote all database object names in generated SQL statements.</td>
</tr>
</tbody>
</table>

## Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddDrop (inherited from TDADumpOptions)</td>
<td>Used to add drop statements to a script before creating statements.</td>
</tr>
<tr>
<td>CompleteInsert (inherited from TDADumpOptions)</td>
<td>Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.</td>
</tr>
<tr>
<td>CreateConstraints</td>
<td>Used to add statements to create table constrains to</td>
</tr>
</tbody>
</table>
See Also

- TPgDumpOptions Class
- TPgDumpOptions Class Members

5.19.1.2.1 CreateConstraints Property

Used to add statements to create table constraints to the dump.

Class

TPgDumpOptions

Syntax

```property CreateConstraints: boolean default True;```

Remarks

If True (the default value), statements to create table constraints are added to the dump. If False, tables are created without constraints.

5.19.2 Types

Types in the PgDump unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgDumpObjects</td>
<td>Represents the set of TPgDumpObject.</td>
</tr>
</tbody>
</table>
5.19.2.1 TPgDumpObjects Set

Represents the set of TPgDumpObject.

Unit
PgDump

Syntax

```
TPgDumpObjects = set of TPgDumpObject;
```

5.19.3 Enumerations

Enumerations in the PgDump unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgDumpMode</td>
<td>Specifies the mode of backup performed by TPgDump.</td>
</tr>
<tr>
<td>TPgDumpObject</td>
<td>Specifies the types of objects that are backed up by PgDump.</td>
</tr>
</tbody>
</table>

5.19.3.1 TPgDumpMode Enumeration

Specifies the mode of backup performed by TPgDump.

Unit
PgDump

Syntax
TPgDumpMode = (dmAll, dmData, dmSchema);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmAll</td>
<td>Backup of schema objects and table data is performed. The default value.</td>
</tr>
<tr>
<td>dmData</td>
<td>Backup of table data only is performed.</td>
</tr>
<tr>
<td>dmSchema</td>
<td>Backup of schema only is performed.</td>
</tr>
</tbody>
</table>

5.19.3.2 TPgDumpObject Enumeration

Specifies the types of objects that are backed up by PgDump.

Unit

PgDump

Syntax

TPgDumpObject = (doDatabase, doDomains, doIndexes, doLanguages, doSchemas, doSequences, doStoredProcs, doTables, doTriggers, doTypes, doViews);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>doDatabase</td>
<td>If set, CREATE DATABASE statement is added to the dump.</td>
</tr>
<tr>
<td>doDomains</td>
<td>Backup of domains is performed.</td>
</tr>
<tr>
<td>doIndexes</td>
<td>Backup of indexes is performed.</td>
</tr>
<tr>
<td>doLanguages</td>
<td>Backup of languages is performed.</td>
</tr>
<tr>
<td>doSchemas</td>
<td>Backup of schemas is performed.</td>
</tr>
<tr>
<td>doSequences</td>
<td>Backup of sequences is performed.</td>
</tr>
<tr>
<td>doStoredProcs</td>
<td>Backup of stored procedures is performed.</td>
</tr>
<tr>
<td>doTables</td>
<td>Backup of tables is performed.</td>
</tr>
<tr>
<td>doTriggers</td>
<td>Backup of triggers is performed.</td>
</tr>
<tr>
<td>doTypes</td>
<td>Backup of types is performed.</td>
</tr>
<tr>
<td>doViews</td>
<td>Backup of views is performed.</td>
</tr>
</tbody>
</table>
5.20 **PgError**

This unit contains the EPgError exception class.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPgError</td>
<td>Represents a PostgreSQL notice, warning, and error messages.</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgSeverity</td>
<td>Specifies the severity of the event occurred.</td>
</tr>
</tbody>
</table>

5.20.1 **Classes**

Classes in the **PgError** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPgError</td>
<td>Represents a PostgreSQL notice, warning, and error messages.</td>
</tr>
</tbody>
</table>

5.20.1.1 **EPgError Class**

Represents a PostgreSQL notice, warning, and error messages.

For a list of all members of this type, see **EPgError** members.
Unit

PgError

Syntax

EPgError = class(EDAError);

Remarks

The EPgError object contains information that the server returns in the form of error, warning, log, notice etc. messages.

Inheritance Hierarchy

EDAError

EPgError

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallStack</td>
<td>Includes a call stack traceback of active procedural language functions and internally-generated queries.</td>
</tr>
<tr>
<td>Component</td>
<td>Contains the component that caused the error.</td>
</tr>
<tr>
<td>DetailMsg</td>
<td>Holds a secondary error message.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>Holds the SQLSTATE code.</td>
</tr>
<tr>
<td>FileName</td>
<td>Used to provide the file name of the source-code location where the error was reported.</td>
</tr>
<tr>
<td>Hint</td>
<td>Provides an optional suggestion on what to do</td>
</tr>
</tbody>
</table>
Properties of the EPgError class.

For a complete list of the EPgError class members, see the EPgError Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallStack</td>
<td>Includes a call stack traceback of active procedural language functions and internally-generated queries.</td>
</tr>
<tr>
<td>Component (inherited from EDAError)</td>
<td>Contains the component that caused the error.</td>
</tr>
<tr>
<td>DetailMsg</td>
<td>Holds a secondary error message.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>Holds the SQLSTATE code.</td>
</tr>
<tr>
<td>FileName</td>
<td>Used to provide the file name of the source-code location where the error was reported.</td>
</tr>
<tr>
<td>Hint</td>
<td>Provides an optional suggestion on what to do about the problem.</td>
</tr>
<tr>
<td>LineNumber</td>
<td>Holds the line number of the source-code location where an error has occurred.</td>
</tr>
<tr>
<td>Position</td>
<td>Holds the error cursor position.</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>ProcedureName</td>
<td>Holds the name of the routine where an error has occurred.</td>
</tr>
<tr>
<td>Severity</td>
<td>Holds the TPgSeverity value.</td>
</tr>
</tbody>
</table>

See Also
- EPgError Class
- EPgError Class Members

5.20.1.1.2.1 CallStack Property

Includes a call stack traceback of active procedural language functions and internally-generated queries.

Class

EPgError

Syntax

```
property callStack: string;
```

Remarks

The CallStack property includes a call stack traceback of active procedural language functions and internally-generated queries. The trace is one entry per line, most recent first.

5.20.1.1.2.2 DetailMsg Property

Holds a secondary error message.

Class

EPgError
Syntax

```property``  DetailMsg: `string`
```

Remarks
The DetailMsg property holds a secondary error message.

Class
EPgError

Syntax

```property``  ErrorCode: `string`
```

Remarks
The ErrorCode property holds the SQLSTATE code.

Class
EPgError

Syntax

```property``  FileName: `string`
```

Remarks
Use the FileName property to get the file name of the source-code location.
5.20.1.1.2.5  Hint Property

Provides an optional suggestion on what to do about the problem.

Class

EPgError

Syntax

```
property Hint: string;
```

Remarks

The Hint property provides a piece of advice (potentially inappropriate) on how the problem can be solved.

5.20.1.1.2.6  LineNumber Property

Holds the line number of the source-code location where an error has occurred.

Class

EPgError

Syntax

```
property LineNumber: integer;
```

Remarks

The LineNumber property holds the line number of the source-code location.
5.20.1.1.2.7 Position Property

Holds the error cursor position.

Class

EPgError

Syntax

```delphi
property Position: integer;
```

Remarks

The Position property holds the error cursor position as 1-based index in the original query string. The position is measured in characters.

5.20.1.1.2.8 ProcedureName Property

Holds the name of the routine where an error has occurred.

Class

EPgError

Syntax

```delphi
property ProcedureName: string;
```

Remarks

The ProcedureName property holds the name of the source-code routine where an error has occurred.

5.20.1.1.2.9 Severity Property

Holds the TPgSeverity value.

Class
EPgError

Syntax

property Severity: TPgSeverity;

Remarks

The Severity property holds the message severity value.

5.20.2 Enumerations

Enumerations in the PgError unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgSeverity</td>
<td>Specifies the severity of the event occurred.</td>
</tr>
</tbody>
</table>

5.20.2.1 TPgSeverity Enumeration

Specifies the severity of the event occurred.

Unit

PgError

Syntax

TPgSeverity = (sError, sFatal, sPanic, sWarning, sNotice, sDebug, sInfo, sLog);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sDebug</td>
<td>Information for developers containing internal server operations.</td>
</tr>
<tr>
<td>sError</td>
<td>Information about a user error.</td>
</tr>
</tbody>
</table>
5.21 **PgLoader**

This unit contains the implementation of the TPgLoader component.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPgLoader</strong></td>
<td>This component serves for loading external data into the database table.</td>
</tr>
<tr>
<td><strong>TPgLoaderColumn</strong></td>
<td>A base class holding a collection of TPgLoaderColumn objects.</td>
</tr>
<tr>
<td><strong>TPgLoaderOptions</strong></td>
<td>This class allows setting up the behaviour of the TPgLoader component.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.21.1 **Classes**

Classes in the **PgLoader** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPgLoader</strong></td>
<td>This component serves for loading external data into the database table.</td>
</tr>
<tr>
<td><strong>TPgLoaderColumn</strong></td>
<td>A base class holding a collection of TPgLoaderColumn objects.</td>
</tr>
</tbody>
</table>
TPgLoader Class

This component serves for loading external data into the database table.

For a list of all members of this type, see TPgLoader members.

Unit

PgLoader

Syntax

TPgLoader = class(TDALoader);

Remarks

TPgLoader allows to load external data into the PostgreSQL database. It uses the COPY command to load data. To specify the name of the loading table set the TableName property. Use the Columns property to access individual columns. Write the OnGetColumnData or OnPutData event handler to read external data and pass it to the database. Call the Load method to start loading data.

Inheritance Hierarchy

TDALoader
  TPgLoader

Properties

TPgLoaderOptions
collection of TPgLoaderColumn objects.
This class allows setting up the behaviour of the TPgLoader component.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BufferSize</strong></td>
<td>Holds the size of the memory buffer used by TPgLoader.</td>
</tr>
<tr>
<td><strong>Columns</strong></td>
<td>Used to access individual columns.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify a connection object and populate the columns connection.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>This class allows setting up the behaviour of the TPgLoader class.</td>
</tr>
<tr>
<td><strong>TableName</strong></td>
<td>Used to specify the name of the loading table set.</td>
</tr>
<tr>
<td><strong>TextMode</strong></td>
<td>Used to load data in the text mode.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CreateColumns</strong></td>
<td>(inherited from <strong>TDALoader</strong>) Creates <strong>TDAColumn</strong> objects for all fields of the table with the same name as <strong>TDALoader.TableName</strong>.</td>
</tr>
<tr>
<td><strong>Load</strong></td>
<td>(inherited from <strong>TDALoader</strong>) Starts loading data.</td>
</tr>
<tr>
<td><strong>LoadFromDataSet</strong></td>
<td>(inherited from <strong>TDALoader</strong>) Loads data from the specified dataset.</td>
</tr>
<tr>
<td><strong>PutColumnData</strong></td>
<td>(inherited from <strong>TDALoader</strong>) Overloaded. Puts the value of individual columns.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnGetColumnData</strong></td>
<td>Used to read external data.</td>
</tr>
<tr>
<td><strong>OnProgress</strong></td>
<td>(inherited from <strong>TDALoader</strong>) Occurs if handling data loading progress of the <strong>TDALoader.LoadFromDataSet</strong> method is needed.</td>
</tr>
<tr>
<td><strong>OnPutData</strong></td>
<td>Used to pass external data to the database.</td>
</tr>
</tbody>
</table>
5.21.1.2 Properties

Properties of the **TPgLoader** class.

For a complete list of the **TPgLoader** class members, see the [TPgLoader Members](#) topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BufferSize</strong></td>
<td>Holds the size of the memory buffer used by TPgLoader.</td>
</tr>
<tr>
<td><strong>TextMode</strong></td>
<td>Used to load data in the text mode.</td>
</tr>
</tbody>
</table>

**Published**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Columns</strong></td>
<td>Used to access individual columns.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Used to specify a connection object and populate the columns connection.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>This class allows setting up the behaviour of the TPgLoader class.</td>
</tr>
<tr>
<td><strong>TableName</strong></td>
<td>Used to specify the name of the loading table set.</td>
</tr>
</tbody>
</table>

See Also

- [TPgLoader Class](#)
- [TPgLoader Class Members](#)
5.21.1.1.2.1 BufferSize Property

Holds the size of the memory buffer used by TPgLoader.

Class
TPgLoader

Syntax

property BufferSize: integer;

Remarks
The BufferSize property contains the size of the memory buffer used by TPgLoader. When the buffer is filled, the loader sends the block of data to the server.

5.21.1.1.2.2 Columns Property

Used to access individual columns.

Class
TPgLoader

Syntax

property Columns: TDAColumns;

Remarks
Use the Columns property to access individual columns.

5.21.1.1.2.3 Connection Property

Used to specify a connection object and populate the columns connection.

Class
TPgLoader
Syntax

**property** Connection: *TPgConnection*;

Remarks

Use the `Connection` property to specify a connection object that will be used to connect to a data store and to populate the columns collection.

© 1997-2021 Devart. All Rights Reserved.

5.21.1.2.4 Options Property

This class allows setting up the behaviour of the `TPgLoader` class.

Class

*TPgLoader*

Syntax

**property** Options: *TPgLoaderOptions*;

© 1997-2021 Devart. All Rights Reserved.

5.21.1.2.5 TableName Property

Used to specify the name of the loading table set.

Class

*TPgLoader*

Syntax

**property** TableName: *string*;

Remarks

Use the `TableName` property to specify the name of the loading table set.

© 1997-2021 Devart. All Rights Reserved.
5.21.1.2.6 TextMode Property

Used to load data in the text mode.

Class

TPgLoader

Syntax

```property` TextMode: boolean;```

5.21.1.3 Events

Events of the TPgLoader class.

For a complete list of the TPgLoader class members, see the TPgLoader Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OnProgress</code> (inherited from TDALoader)</td>
<td>Occurs if handling data loading progress of the TDALoader.LoadFromData Set method is needed.</td>
</tr>
</tbody>
</table>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OnGetColumnData</code></td>
<td>Used to read external data.</td>
</tr>
<tr>
<td><code>OnPutData</code></td>
<td>Used to pass external data to the database.</td>
</tr>
</tbody>
</table>

See Also

- TPgLoader Class
- TPgLoader Class Members

© 1997-2021

Devart. All Rights Reserved.
5.21.1.3.1  OnGetColumnData Event

Used to read external data.

**Class**  
*TPgLoader*

**Syntax**

```
property OnGetColumnData: TGetColumnDataEvent;
```

**Remarks**

Write the OnGetColumnData event handler to read external data.

© 1997-2021  
Devart. All Rights Reserved.

5.21.1.3.2  OnPutData Event

Used to pass external data to the database.

**Class**  
*TPgLoader*

**Syntax**

```
property OnPutData: TDAPutDataEvent;
```

**Remarks**

Write the OnPutData event handler to pass external data to the database.

© 1997-2021  
Devart. All Rights Reserved.

5.21.1.2  *TPgLoaderColumn Class*

A base class holding a collection of *TPgLoaderColumn* objects.

For a list of all members of this type, see *TPgLoaderColumn* members.

**Unit**  
*PgLoader*
Syntax

```cpp
TPgLoaderColumn = class(TDAColumn);
```

Remarks

Each TPgLoader uses TPgLoaderColumn to maintain a collection of TPgLoaderColumn objects. TPgLoaderColumn object represents the attributes for column loading. Every TPgLoaderColumn object corresponds to one of the table fields with the same name as its Name property.

To create columns at design time use column editor of TPgLoader component.

Inheritance Hierarchy

- `TDAColumn`
- `TPgLoaderColumn`

See Also

- `TPgLoader`

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FieldType</code> (inherited from <code>TDAColumn</code>)</td>
<td>Used to specify the types of values that will be loaded.</td>
</tr>
<tr>
<td><code>Name</code> (inherited from <code>TDAColumn</code>)</td>
<td>Used to specify the field name of loading table.</td>
</tr>
<tr>
<td><code>RowTypeName</code></td>
<td>Holds the name of a composite type that corresponds to this field.</td>
</tr>
</tbody>
</table>
5.21.1.2.2 Properties

Properties of the TPgLoaderColumn class.

For a complete list of the TPgLoaderColumn class members, see the TPgLoaderColumn Members topic.

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FieldType (inherited from TDAColumn)</td>
<td>Used to specify the types of values that will be loaded.</td>
</tr>
<tr>
<td>Name (inherited from TDAColumn)</td>
<td>Used to specify the field name of loading table.</td>
</tr>
<tr>
<td>RowTypeName</td>
<td>Holds the name of a composite type that corresponds to this field.</td>
</tr>
</tbody>
</table>

See Also
- TPgLoaderColumn Class
- TPgLoaderColumn Class Members

5.21.1.2.2.1 RowTypeName Property

Holds the name of a composite type that corresponds to this field.

Class

TPgLoaderColumn

Syntax

```property` RowTypeName: `string;```

Remarks

The RowTypeName property holds the name of a composite type that corresponds to this field. If a field is of the composite (ROW) data type, assign the name of the type to this property. This is required for loading values of composite type in the binary mode.
Reserved.

5.21.1.3 TPgLoaderOptions Class

This class allows setting up the behaviour of the TPgLoader component.

For a list of all members of this type, see TPgLoaderOptions members.

Unit

PgLoader

Syntax

TPgLoaderOptions = class(TDALoaderOptions);

Inheritance Hierarchy

TDALoaderOptions
TPgLoaderOptions

5.21.1.3.1 Members

TPgLoaderOptions class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufferSize</td>
<td>Holds the size of the memory buffer used by TPgLoader.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used to quote columns names in the INSERT statement used for loading data.</td>
</tr>
<tr>
<td>TextMode</td>
<td>Used to load data in the text mode.</td>
</tr>
<tr>
<td>UseBlankValues</td>
<td>Forces PgDAC to fill the buffer with null values after loading a row to the database.</td>
</tr>
</tbody>
</table>
Properties of the **TPgLoaderOptions** class.

For a complete list of the **TPgLoaderOptions** class members, see the [TPgLoaderOptions Members](#) topic.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufferSize</td>
<td>Holds the size of the memory buffer used by TPgLoader.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used to quote column names in the <code>INSERT</code> statement used for loading data.</td>
</tr>
<tr>
<td>TextMode</td>
<td>Used to load data in the text mode.</td>
</tr>
<tr>
<td>UseBlankValues</td>
<td>Forces PgDAC to fill the buffer with null values after loading a row to the database.</td>
</tr>
</tbody>
</table>

**See Also**
- [TPgLoaderOptions Class](#)
- [TPgLoaderOptions Class Members](#)

5.21.1.3.2.1 BufferSize Property

Holds the size of the memory buffer used by TPgLoader.
### BufferSize Property

**property** BufferSize: integer **default** DefaultBufferSize;

**Remarks**

The BufferSize property contains the size of the memory buffer used by TPgLoader. When buffer is filled, the loader sends block of data to the server.

### QuoteNames Property

**Remarks**

Used to quote columns names in the INSERT statement used for loading data.

**Class**

TPgLoaderOptions

**Syntax**

```pascal
property QuoteNames: boolean;
```

### TextMode Property

**Remarks**

Use the QuoteNames property to specify whether the column names in the INSERT statement will be quoted.

When the property value is *True*, then the column names will be quoted. When the property value is *False* then the column names will not be quoted.

The default value is *False*.

**Class**

TPgLoaderOptions

**Syntax**

```pascal
property TextMode: boolean **default** False;
```
Remarks

Use the TextMode property to load data in the text mode.

TPgLoader supports two load modes: text and binary. By default the binary mode is used for a connection with 3.0 protocol. Set TextMode property to True to force text mode. In binary mode TPgLoader may work slightly faster but some data type are not supported in this mode. In text mode you can load data to columns with any PostgreSQL data type.

5.21.1.3.2.4 UseBlankValues Property

Forces PgDAC to fill the buffer with null values after loading a row to the database.

Class

TPgLoaderOptions

Syntax

@property UseBlankValues: boolean;

Remarks

Used to force PgDAC to fill the buffer with null values after loading a row to the database.

5.22 PgObjects

This unit contains classes for PostgreSQL specific data types.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomPgTimeStamp</td>
<td>A base class for the TPgDate, TPgTime, and TPgTimeStamp classes.</td>
</tr>
<tr>
<td>TPgAttribute</td>
<td>A class holding the description of an attribute of PostgreSQL composite</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TPgBox</strong></td>
<td>A class for working with PostgreSQL BOX data type.</td>
</tr>
<tr>
<td><strong>TPgCircle</strong></td>
<td>A class for working with PostgreSQL CIRCLE data type.</td>
</tr>
<tr>
<td><strong>TPgDate</strong></td>
<td>A class for working with PostgreSQL DATE data type.</td>
</tr>
<tr>
<td><strong>TPgGeometric</strong></td>
<td>A base class for classes that work with geometric data types.</td>
</tr>
<tr>
<td><strong>TPgInterval</strong></td>
<td>A class providing support of PostgreSQL INTERVAL datatype.</td>
</tr>
<tr>
<td><strong>TPgLSeg</strong></td>
<td>A class for working with PostgreSQL LSEG (line segment) data type.</td>
</tr>
<tr>
<td><strong>TPgPath</strong></td>
<td>A class for working with PostgreSQL PATH data type.</td>
</tr>
<tr>
<td><strong>TPgPoint</strong></td>
<td>A class for working with PostgreSQL POINT data type.</td>
</tr>
<tr>
<td><strong>TPgPointsArray</strong></td>
<td>A base class for working with geometric data types.</td>
</tr>
<tr>
<td><strong>TPgPolygon</strong></td>
<td>A class for working with PostgreSQL POLYGON data type.</td>
</tr>
<tr>
<td><strong>TPgRefCursor</strong></td>
<td>A class for working with REFCURSOR values.</td>
</tr>
<tr>
<td><strong>TPgRow</strong></td>
<td>A class for working with PostgreSQL composite (ROW) data types.</td>
</tr>
<tr>
<td><strong>TPgRowType</strong></td>
<td>A class holding the description of a PostgreSQL composite (ROW) type.</td>
</tr>
<tr>
<td><strong>TPgSQLLargeObject</strong></td>
<td>A class for working with PostgreSQL large objects.</td>
</tr>
<tr>
<td><strong>TPgTime</strong></td>
<td>A class for working with PostgreSQL TIME and TIME WITH TIMEZONE data type.</td>
</tr>
<tr>
<td><strong>TPgTimeStamp</strong></td>
<td>A class for working with PostgreSQL TIMESTAMP</td>
</tr>
</tbody>
</table>
Classes in the **PgObjects** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomPgTimeStamp</td>
<td>A base class for the TPgDate, TPgTime, and TPgTimeStamp classes.</td>
</tr>
<tr>
<td>TPgAttribute</td>
<td>A class holding the description of an attribute of PostgreSQL composite type.</td>
</tr>
<tr>
<td>TPgBox</td>
<td>A class for working with PostgreSQL BOX data type.</td>
</tr>
<tr>
<td>TPgCircle</td>
<td>A class for working with PostgreSQL CIRCLE data type.</td>
</tr>
<tr>
<td>TPgDate</td>
<td>A class for working with PostgreSQL DATE data type.</td>
</tr>
<tr>
<td>TPgGeometric</td>
<td>A base class for classes that work with geometric data types.</td>
</tr>
<tr>
<td>TPgInterval</td>
<td>A class providing support of PostgreSQL INTERVAL datatype.</td>
</tr>
<tr>
<td>TPgLSeg</td>
<td>A class for working with PostgreSQL LSEG (line segment) data type.</td>
</tr>
<tr>
<td>TPgPath</td>
<td>A class for working with PostgreSQL PATH data type.</td>
</tr>
<tr>
<td>TPgPoint</td>
<td>A class for working with PostgreSQL POINT data type.</td>
</tr>
</tbody>
</table>
### 5.22.1.1 TCustomPgTimeStamp Class

A base class for the TPgDate, TPgTime, and TPgTimeStamp classes.

For a list of all members of this type, see [TCustomPgTimeStamp members](#).

#### Unit

`PgObjects`

#### Syntax

```
TCustomPgTimeStamp = class(TSharedObject);
```

#### Remarks

The TCustomPgTimeStamp class is a base class for the TPgDate, TPgTime, and TPgTimeStamp classes.
Inheritance Hierarchy

TSharedObject
TCustomPgTimeStamp

© 1997-2021
Devart. All Rights Reserved.

5.22.1.1.1 Members

**TCustomPgTimeStamp** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsDateTime</td>
<td>Used to get and set the timestamp value as TDateTime.</td>
</tr>
<tr>
<td>AsSQLTime Stamp</td>
<td>Used to get and set the timestamp value as TSQLTime Stamp record.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to get and set the timestamp value as string.</td>
</tr>
<tr>
<td>Days</td>
<td>Holds the date part of timestamp.</td>
</tr>
<tr>
<td>HasTimeZone</td>
<td>Used to specify whether the timestamp object has time zone.</td>
</tr>
<tr>
<td>IsInfinity</td>
<td>Determines if a timestamp object has +/-infinity value.</td>
</tr>
<tr>
<td>IsNegInfinity</td>
<td>Determines if a timestamp object has -infinity value.</td>
</tr>
<tr>
<td>IsPosInfinity</td>
<td>Determines if a timestamp object has +infinity value.</td>
</tr>
<tr>
<td>RefCount (inherited from TSharedObject)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Ticks</td>
<td>Holds the time part of timestamp in microseconds.</td>
</tr>
<tr>
<td>TimeZone Offset</td>
<td>Specifies the time zone offset of a timestamp object (in seconds).</td>
</tr>
</tbody>
</table>
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from <strong>TSharedObject</strong>)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Assigns a value from another TCustomPgTimeStamp object to this object.</td>
</tr>
<tr>
<td>Compare</td>
<td>Compares two TCustomPgTimeStamp objects.</td>
</tr>
<tr>
<td>DecodeDate</td>
<td>Provides year, month, and day from the timestamp object.</td>
</tr>
<tr>
<td>DecodeDateTime</td>
<td>Provides the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.</td>
</tr>
<tr>
<td>DecodeTime</td>
<td>Provides hour, minute, second, and microsecond from the timestamp object.</td>
</tr>
<tr>
<td>EncodeDate</td>
<td>Sets year, month, and day in the timestamp object.</td>
</tr>
<tr>
<td>EncodeDateTime</td>
<td>Sets the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.</td>
</tr>
<tr>
<td>EncodeTime</td>
<td>Sets hour, minute, second, and microsecond in the timestamp object.</td>
</tr>
<tr>
<td>Release (inherited from <strong>TSharedObject</strong>)</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.22.1.1.2 Properties

Properties of the **TCustomPgTimeStamp** class.

For a complete list of the **TCustomPgTimeStamp** class members, see the
TCustomPgTimeStamp Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsDateTime</td>
<td>Used to get and set the timestamp value as TDateTime.</td>
</tr>
<tr>
<td>AsSQLTimeStamp</td>
<td>Used to get and set the timestamp value as TSQLTimeStamp record.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to get and set the timestamp value as string.</td>
</tr>
<tr>
<td>Days</td>
<td>Holds the date part of timestamp.</td>
</tr>
<tr>
<td>HasTimeZone</td>
<td>Used to specify whether the timestamp object has time zone.</td>
</tr>
<tr>
<td>IsInfinity</td>
<td>Determines if a timestamp object has +/-infinity value.</td>
</tr>
<tr>
<td>IsNegInfinity</td>
<td>Determines if a timestamp object has -infinity value.</td>
</tr>
<tr>
<td>IsPosInfinity</td>
<td>Determines if a timestamp object has +infinity value.</td>
</tr>
<tr>
<td>RefCount (inherited from TSharedObject)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Ticks</td>
<td>Holds the time part of timestamp in microseconds.</td>
</tr>
<tr>
<td>TimeZoneOffset</td>
<td>Specifies the time zone offset of a timestamp object (in seconds).</td>
</tr>
</tbody>
</table>

See Also
- TCustomPgTimeStamp Class
- TCustomPgTimeStamp Class Members

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
5.22.1.1.2.1 AsDateTime Property

Used to get and set the timestamp value as TDateTime.

Class
TCustomPgTimeStamp

Syntax

```pascal
property AsDateTime: TDateTime;
```

Remarks
Use the AsDateTime property to get and set the timestamp value as TDateTime.

5.22.1.1.2.2 AsSQLTimeStamp Property

Used to get and set the timestamp value as TSQLTimeStamp record.

Class
TCustomPgTimeStamp

Syntax

```pascal
property AsSQLTimeStamp: TSQLTimeStamp;
```

Remarks
Use the AsString property to get and set the timestamp value as TSQLTimeStamp record.

5.22.1.1.2.3 AsString Property

Used to get and set the timestamp value as string.

Class
TCustomPgTimeStamp
Syntax

```property AsString: string;```

Remarks

Use the `AsString` property to get and set the timestamp value as string.

© 1997-2021 Devart. All Rights Reserved.

5.22.1.1.2.4 Days Property

Holds the date part of timestamp.

Class

`TCustomPgTimeStamp`

Syntax

```property Days: integer;```

Remarks

The `Days` property holds the date part of timestamp as a number of days since 01-01-2000.

See Also

- `DecodeDate`
- `EncodeDate`

© 1997-2021 Devart. All Rights Reserved.

5.22.1.1.2.5 HasTimeZone Property

Used to specify whether the timestamp object has time zone.

Class

`TCustomPgTimeStamp`

Syntax

```property HasTimeZone: boolean;```
Remarks

Use the HasTimeZone property to specify whether the timestamp object has time zone.

5.22.1.1.2.6  IsInfinity Property

Determines if a timestamp object has +/-infinity value.

Class

TCustomPgTimeStamp

Syntax

```delphi
property IsInfinity: boolean;
```

Remarks

Use the IsInfinity property to determine if a timestamp object has +/-infinity value. The IsInfinity property is True, if timestamp value is +infinity or -infinity, and False, if the value is not infinity.

5.22.1.1.2.7  IsNegInfinity Property

Determines if a timestamp object has -infinity value.

Class

TCustomPgTimeStamp

Syntax

```delphi
property IsNegInfinity: boolean;
```

Remarks

Use the IsNegInfinity property to determine if a timestamp object has -infinity value. The IsNegInfinity property is True, if timestamp value is -infinity.
5.22.1.1.2.8  IsPosInfinity Property

Determines if a timestamp object has +infinity value.

Class

TCustomPgTimeStamp

Syntax

property IsPosInfinity: boolean;

Remarks

Use the IsPosInfinity property to determine if a timestamp object has +infinity value. The IsPosInfinity property is True, if timestamp value is +infinity.

5.22.1.1.2.9  Ticks Property

Holds the time part of timestamp in microseconds.

Class

TCustomPgTimeStamp

Syntax

property Ticks: int64;

Remarks

The Ticks property holds the time part of timestamp in microseconds.

See Also

• DecodeTime
• EncodeTime

© 1997-2021
Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback

© 2021 Devart
5.22.1.1.10 TimeZoneOffset Property

Specifies the time zone offset of a timestamp object (in seconds).

Class

TCustomPgTimeStamp

Syntax

property TimeZoneOffset: integer;

Remarks

Use the TimeZoneOffset property to specify the time zone offset of a timestamp object (in seconds).

For values of TIMESTAMP WITH TIMEZONE read this property to get the offset of the timestamp value from Universal Time (UTC).

The value of TimeZoneOffset is specified in seconds.

© 1997-2021 Devart. All Rights Reserved.

5.22.1.1.3 Methods

Methods of the TCustomPgTimeStamp class.

For a complete list of the TCustomPgTimeStamp class members, see the TCustomPgTimeStamp Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>(inherited from TSharedObject) Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Assigns a value from another TCustomPgTimeStamp object to this object.</td>
</tr>
<tr>
<td>Compare</td>
<td>Compares two TCustomPgTimeStamp objects.</td>
</tr>
</tbody>
</table>
### DecodeDate
Provides year, month, and day from the timestamp object.

### DecodeDateTime
Provides the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.

### DecodeTime
Provides hour, minute, second, and microsecond from the timestamp object.

### EncodeDate
Sets year, month, and day in the timestamp object.

### EncodeDateTime
Sets the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.

### EncodeTime
Sets hour, minute, second, and microsecond in the timestamp object.

### Release
(inherited from TSharedObject)
Decrement the reference count.

---

**See Also**
- [TCustomPgTimeStamp Class](#)
- [TCustomPgTimeStamp Class Members](#)

© 1997-2021 Devart. All Rights Reserved.

5.22.1.3.1 Assign Method

Assigns a value from another TCustomPgTimeStamp object to this object.

**Class**

TCustomPgTimeStamp

**Syntax**

```plaintext
procedure Assign(Source: TCustomPgTimeStamp); virtual;
```

**Parameters**

**Source**
Holds the TCustomPgTimeStamp object to assign a value from.
Remarks

Call the Assign property to assign a value from another TCustomPgTime_stamp object to this object.

5.22.1.1.3.2 Compare Method

Compares two TCustomPgTime_stamp objects.

Class

TCustomPgTime_stamp

Syntax

function Compare(Value: TCustomPgTime_stamp): integer; virtual;
abstract;

Parameters

Value
Holds a TCustomPgTime_stamp object to compare with.

Remarks

Call the Compare method to compare two TCustomPgTime_stamp objects.

If the current object represent date and time that is greater than Value, Compare returns a positive number.

If the current object represent date and time that is lower than Value, Compare returns a negative number.

If the current object equals to Value, Compare returns 0.

5.22.1.1.3.3 DecodeDate Method

Provides year, month, and day from the timestamp object.

Class
**TCustomPgTimeStamp**

**Syntax**

```plaintext
procedure DecodeDate(var Year: integer; var Month: integer; var Day: integer);
```

**Parameters**

- **Year**
  - Holds the year.
- **Month**
  - Holds the month.
- **Day**
  - Holds the day.

**Remarks**

Call the `DecodeDate` method to get year, month, and day from the timestamp object.

---

5.22.1.3.4  DecodeDateTime Method

Provides the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.

**Class**

**TCustomPgTimeStamp**

**Syntax**

```plaintext
procedure DecodeDateTime(var Year: integer; var Month: integer; var
Day: integer; var Hour: integer; var Minute: integer; var
Second: integer; var Microsecond: integer);
```

**Parameters**

- **Year**
  - Holds the year.
- **Month**
  - Holds the month.
- **Day**
  - Holds the day.
5.22.1.3.5 DecodeTime Method

Provides hour, minute, second, and microsecond from the timestamp object.

Class

TCustomPgTimeStamp

Syntax

procedure DecodeTime(var Hour: integer; var Minute: integer; var Second: integer; var Microsecond: integer);

Parameters

Hour
Holds the hour.

Minute
Holds the minute.

Second
Holds the second.

Microsecond
Holds the microsecond.

Remarks

Call the DecodeTime method to get hour, minute, second, and microsecond from the timestamp object.
5.22.1.3.6 EncodeDate Method

Sets year, month, and day in the timestamp object.

Class

TCustomPgTimeStamp

Syntax

procedure EncodeDate(Year: integer; Month: integer; Day: integer);

Parameters

Year
   Holds the year.
Month
   Holds the month.
Day
   Holds the day.

Remarks

Call the EncodeDate property to set year, month, and day in the timestamp object.

5.22.1.3.7 EncodeDateTime Method

Sets the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.

Class

TCustomPgTimeStamp

Syntax

procedure EncodeDateTime(Year: integer; Month: integer; Day: integer; Hour: integer; Minute: integer; Second: integer; Microsecond: integer);
Parameters

Year
  Holds the year.
Month
  Holds the month.
Day
  Holds the day.
Hour
  Holds the hour.
Minute
  Holds the minute.
Second
  Holds the second.
Microsecond
  Holds the microsecond.

Remarks

Call the EncodeDateTime method to set the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.

Class

TCustomPgTimeStamp

Syntax

procedure EncodeTime(Hour: integer; Minute: integer; Second: integer; Microsecond: integer);

Parameters

Hour
  Holds the hour.
Minute
  Holds the minute.
Second
Holds the second.

*Microsecond*
Holds the microsecond.

**Remarks**

Call the `EncodeTime` method to set hour, minute, second, and microsecond in the timestamp object.

```
© 1997-2021
Devart. All Rights Reserved.
```

### 5.22.1.2 TPgAttribute Class

A class holding the description of an attribute of PostgreSQL composite type.

For a list of all members of this type, see [TPgAttribute](#) members.

**Unit**

*pgobjects*

**Syntax**

```
TPgAttribute = class(TAttribute);
```

**Remarks**

The TPgAttribute class holds the description of an attribute of PostgreSQL composite type. You can use TPgRowType.Attributes to access individual attributes.

**Inheritance Hierarchy**

- [TAttribute](#)
- [TPgAttribute](#)

© 1997-2021
Devart. All Rights Reserved.

### 5.22.1.2.1 Members

**TPgAttribute** class overview.

**Properties**
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeNo</td>
<td>Returns an attribute’s ordinal position in object.</td>
</tr>
<tr>
<td>DataSize</td>
<td>Returns the size of an attribute value in internal representation.</td>
</tr>
<tr>
<td>DataType</td>
<td>Returns the type of data that was assigned to the Attribute.</td>
</tr>
<tr>
<td>Length</td>
<td>Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Returns a TObjectType object for an object attribute.</td>
</tr>
<tr>
<td>Offset</td>
<td>Returns an offset of the attribute value in internal representation.</td>
</tr>
<tr>
<td>Owner</td>
<td>Indicates TObjectType that uses the attribute to represent one of its attributes.</td>
</tr>
<tr>
<td>Scale</td>
<td>Returns the scale of dtFloat and dtInteger attributes.</td>
</tr>
<tr>
<td>Size</td>
<td>Returns the size of an attribute value in external representation.</td>
</tr>
</tbody>
</table>

5.22.1.3 **TPgBox Class**

A class for working with PostgreSQL BOX data type.

For a list of all members of this type, see [TPgBox members](#).

**Unit**

*PgObjects*

**Syntax**

```pascal
TPgBox = class(TPgPointsArray);
```
Remarks
The TPgBox class is used to work with PostgreSQL BOX data type.

Inheritance Hierarchy

- TSharedObject
  - TPgGeometric
    - TPgPointsArray
    - TPgBox

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString (inherited from TPgGeometric)</td>
<td>Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td>LowerLeft</td>
<td>Holds the lower-left corner of the box.</td>
</tr>
<tr>
<td>RefCount (inherited from TSharedObject)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>UpperRight</td>
<td>Holds the upper-right corner of the box.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign (inherited from TPgGeometric)</td>
<td>Assigns a value from another TPgGeometric object to this object.</td>
</tr>
<tr>
<td>Release (inherited from TSharedObject)</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>
5.22.1.3.2 Properties

Properties of the **TPgBox** class.

For a complete list of the **TPgBox** class members, see the [TPgBox Members](#) topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsString</strong></td>
<td>(inherited from <strong>TPgGeometric</strong>) Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td><strong>LowerLeft</strong></td>
<td>Holds the lower-left corner of the box.</td>
</tr>
<tr>
<td><strong>RefCount</strong></td>
<td>(inherited from <strong>TSharedObject</strong>) Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td><strong>UpperRight</strong></td>
<td>Holds the upper-right corner of the box.</td>
</tr>
</tbody>
</table>

See Also
- **TPgBox Class**
- **TPgBox Class Members**

5.22.1.3.2.1 LowerLeft Property

Holds the lower-left corner of the box.

**Class**

**TPgBox**

**Syntax**

```
property LowerLeft: TPgPoint;
```  

**Remarks**

The LowerLeft property holds the lower-left corner of the box.
5.22.1.3.2.2 UpperRight Property

Holds the upper-right corner of the box.

Class

TPgBox

Syntax

```
property UpperRight: TPgPoint;
```

Remarks

The UpperRight property holds the upper-right corner of the box.

5.22.1.4 TPgCircle Class

A class for working with PostgreSQL CIRCLE data type.

For a list of all members of this type, see TPgCircle members.

Unit

PgObjects

Syntax

```
TPgCircle = class(TPgGeometric);
```

Remarks

The TPgCircle class is used to work with PostgreSQL CIRCLE data type.

Inheritance Hierarchy

```
TSharedObject
  TPgGeometric
  TPgCircle
```
5.22.1.4.1 Members

**TPgCircle** class overview.

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsString</strong> (inherited from <strong>TPgGeometric</strong>)</td>
<td>Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td><strong>Center</strong></td>
<td>Holds the center point of the circle.</td>
</tr>
<tr>
<td><strong>Radius</strong></td>
<td>Holds the radius of the circle.</td>
</tr>
<tr>
<td><strong>RefCount</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddRef</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td><strong>Assign</strong> (inherited from <strong>TPgGeometric</strong>)</td>
<td>Assigns a value from another TPgGeometric object to this object.</td>
</tr>
<tr>
<td><strong>Release</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

Properties of the **TPgCircle** class.

For a complete list of the **TPgCircle** class members, see the [TPgCircle Members](#) topic.

Public
### Name | Description
--- | ---
**AsString** (inherited from **TPgGeometric**) | Used to get and set the geometric value as string.
**Center** | Holds the center point of the circle.
**Radius** | Holds the radius of the circle.
**RefCount** (inherited from **TSharedObject**) | Used to return the count of reference to a TSharedObject object.

See Also
- **TPgCircle Class**
- **TPgCircle Class Members**

#### 5.22.1.4.2.1 Center Property

Holds the center point of the circle.

**Class**

**TPgCircle**

**Syntax**

```property
Center: TPgPoint;
```

**Remarks**

The Center property holds the center point of the circle.
Syntax

```property
Radius: Double;
```

Remarks

The Radius property holds the radius of the circle.

© 1997-2021 Devart. All Rights Reserved.

5.22.1.5 TPgDate Class

A class for working with PostgreSQL DATE data type.

For a list of all members of this type, see TPgDate members.

Unit

`PgObjects`

Syntax

```text
TPgDate = class(TCustomPgTimeStamp);
```

Remarks

The TPgDate class is used to work with PostgreSQL DATE data type.

Inheritance Hierarchy

```
TSharedObject
  TCustomPgTimeStamp
    TPgDate
```

© 1997-2021 Devart. All Rights Reserved.

5.22.1.5.1 Members

**TPgDate** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsDateTime</td>
<td>(inherited from TCustomPgTimeStamp) Used to get and set the timestamp value as TDateTime.</td>
</tr>
<tr>
<td>AsSQLTimeStamp</td>
<td>(inherited from TCustomPgTimeStamp) Used to get and set the timestamp value as TSQLTimeStamp record.</td>
</tr>
<tr>
<td>AsString</td>
<td>(inherited from TCustomPgTimeStamp) Used to get and set the timestamp value as string.</td>
</tr>
<tr>
<td>Days</td>
<td>(inherited from TCustomPgTimeStamp) Holds the date part of timestamp.</td>
</tr>
<tr>
<td>HasTimeZone</td>
<td>(inherited from TCustomPgTimeStamp) Used to specify whether the timestamp object has time zone.</td>
</tr>
<tr>
<td>IsInfinity</td>
<td>(inherited from TCustomPgTimeStamp) Determines if a timestamp object has +/-infinity value.</td>
</tr>
<tr>
<td>IsNegInfinity</td>
<td>(inherited from TCustomPgTimeStamp) Determines if a timestamp object has -infinity value.</td>
</tr>
<tr>
<td>IsPosInfinity</td>
<td>(inherited from TCustomPgTimeStamp) Determines if a timestamp object has +infinity value.</td>
</tr>
<tr>
<td>RefCount</td>
<td>(inherited from TSharedObject) Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Ticks</td>
<td>(inherited from TCustomPgTimeStamp) Holds the time part of timestamp in microseconds.</td>
</tr>
<tr>
<td>TimeZoneOffset</td>
<td>(inherited from TCustomPgTimeStamp) Specifies the time zone offset of a timestamp object (in seconds).</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>(inherited from TSharedObject) Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>(inherited from TCustomPgTimeStamp) Assigns a value from another TCustomPgTimeStamp object to this object.</td>
</tr>
<tr>
<td>Compare</td>
<td>(inherited from TCustomPgTimeStamp) Compares two TCustomPgTimeStamp objects.</td>
</tr>
<tr>
<td>DecodeDate</td>
<td>(inherited from TCustomPgTimeStamp) Provides year, month, and day from the timestamp</td>
</tr>
</tbody>
</table>
TPgGeometric Class

A base class for classes that work with geometric data types.

For a list of all members of this type, see TPgGeometric members.

Unit
PgObjects

Syntax

TPgGeometric = class(TSharedObject);

Remarks

The TPgGeometric class is a class for classes that work with geometric data types.

Inheritance Hierarchy
TSharedObject
TPgGeometric
5.22.1.6.1 Members

**TPgGeometric** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td>RefCount</td>
<td>(inherited from <strong>TSharedObject</strong>) Used to return the count of reference to</td>
</tr>
<tr>
<td></td>
<td>a <strong>TSharedObject</strong> object.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>(inherited from <strong>TSharedObject</strong>) Increments the reference count for the</td>
</tr>
<tr>
<td></td>
<td>number of references dependent on the <strong>TSharedObject</strong> object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Assigns a value from another <strong>TPgGeometric</strong> object to this object.</td>
</tr>
<tr>
<td>Release</td>
<td>(inherited from <strong>TSharedObject</strong>) Decrements the reference count.</td>
</tr>
</tbody>
</table>

5.22.1.6.2 Properties

Properties of the **TPgGeometric** class.

For a complete list of the **TPgGeometric** class members, see the **TPgGeometric Members** topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

### AsString

**Used to get and set the geometric value as string.**

### RefCount (inherited from TSharedObject)

**Used to return the count of reference to a TSharedObject object.**

---

#### See Also

- [TPgGeometric Class](#)
- [TPgGeometric Class Members](#)

---

#### 5.22.1.6.2.1 AsString Property

Used to get and set the geometric value as string.

**Class**

TPgGeometric

**Syntax**

```
property AsString: string;
```

**Remarks**

Use the AsString property to get and set the geometric value as string.

---

#### 5.22.1.6.3 Methods

Methods of the TPgGeometric class.

For a complete list of the TPgGeometric class members, see the [TPgGeometric Members](#) topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved. 

[Request Support] [DAC Forum] [Provide Feedback]
Assigns a value from another TPgGeometric object to this object.

Class

TPgGeometric

Syntax

```pascal
procedure Assign(Source: TPgGeometric); virtual;
```

Parameters

Source

Holds a TPgGeometric object to assign the value from.

Remarks

Call the Assign property to assign a value from another TPgGeometric object to this object.

5.22.1.7 TPgInterval Class

A class providing support of PostgreSQL INTERVAL datatype.

For a list of all members of this type, see TPgInterval members.
Unit
PgObjects

Syntax

```delphi
TPgInterval = class(TSharedObject);
```

Remarks

The TPgInterval class is used to work with PostgreSQL INTERVAL data type.

Inheritance Hierarchy

```delphi
TSharedObject
TPgInterval
```

© 1997-2021
Devart. All Rights Reserved.

5.22.1.7.1 Members

**TPgInterval** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to get and set the INTERVAL value as string.</td>
</tr>
<tr>
<td>Days</td>
<td>Used to get the days part of an interval.</td>
</tr>
<tr>
<td>MonthsFull</td>
<td>Used to get the months part of an interval.</td>
</tr>
<tr>
<td>RefCount</td>
<td>(inherited from TSharedObject) Used to return the count of reference to a</td>
</tr>
<tr>
<td></td>
<td>TSharedObject object.</td>
</tr>
<tr>
<td>SecondsFull</td>
<td>Used to get the time part of an interval represented in seconds.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
AddRef (inherited from TSharedObject) | Increments the reference count for the number of references dependent on the TSharedObject object.
---|---
Assign | Assigns a value from another TPgInterval object to this object.
Compare | Compares two TPgInterval objects.
DecodeInterval | Provides the value of interval as years, months, days, hours, minutes, seconds, and microseconds.
EncodeInterval | Sets the value of interval as years, months, days, hours, minutes, seconds, and microseconds.
Release (inherited from TSharedObject) | Decrements the reference count.

5.22.1.7.2 Properties

Properties of the TPgInterval class.

For a complete list of the TPgInterval class members, see the TPgInterval Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to get and set the INTERVAL value as string.</td>
</tr>
<tr>
<td>Days</td>
<td>Used to get the days part of an interval.</td>
</tr>
<tr>
<td>MonthsFull</td>
<td>Used to get the months part of an interval.</td>
</tr>
<tr>
<td>RefCount (inherited from TSharedObject)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>SecondsFull</td>
<td>Used to get the time part of an interval represented in seconds.</td>
</tr>
</tbody>
</table>
5.22.1.7.2.1 AsString Property

Used to get and set the INTERVAL value as string.

Class

TPgInterval

Syntax

propertygetAsString: string;

Remarks
Use the AsString property to get and set the INTERVAL value as string.

5.22.1.7.2.2 Days Property

Used to get the days part of an interval.

Class

TPgInterval

Syntax

property Days: integer;

Remarks
Use the Days property to get the days part of an interval.
5.22.1.7.2.3 MonthsFull Property

Used to get the months part of an interval.

Class
TPgInterval

Syntax

property MonthsFull: integer;

Remarks
Use the MonthsFull property to get the months part of an interval.

The months part of an interval that includes months and years converted to months.

5.22.1.7.2.4 SecondsFull Property

Used to get the time part of an interval represented in seconds.

Class
TPgInterval

Syntax

property SecondsFull: double;

Remarks
Use the SecondsFull property to get the time part of an interval represented in seconds.

5.22.1.7.3 Methods

Methods of the TPgInterval class.

For a complete list of the TPgInterval class members, see the TPgInterval Members topic.
### Assign Method

Assigns a value from another TPgInterval object to this object.

**Class**

**TPgInterval**

**Syntax**

```plaintext
procedure Assign(Source: TPgInterval);
```

**Parameters**

- **Source**
  Holds the TPgInterval object to assign value from.

---

**Name** | **Description**
--- | ---
AddRef (inherited from **TSharedObject**) | Increments the reference count for the number of references dependent on the TSharedObject object.

Assign | Assigns a value from another TPgInterval object to this object.

Compare | Compares two TPgInterval objects.

DecodeInterval | Provides the value of interval as years, months, days, hours, minutes, seconds, and microseconds.

EncodeInterval | Sets the value of interval as years, months, days, hours, minutes, seconds, and microseconds.

Release (inherited from **TSharedObject**) | Decrements the reference count.

---

**See Also**

- **TPgInterval Class**
- **TPgInterval Class Members**

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

© 2021 Devart
Remarks

Call the Assign method to assign a value from another TPgInterval object to this object.

Class

TPgInterval

Syntax

```delphi
function Compare(Value: TPgInterval): integer;
```

Parameters

**Value**

Holds a TPgInterval object to compare with.

Remarks

Call the Compare method to compare two TPgInterval objects.

If the current object represents an interval that is longer than Value, Compare returns a positive number. If the current object represents an interval that is shorter than Value, Compare returns a negative number. If the current object equals to Value, Compare returns 0.

Internally INTERVAL values are stored as months, days, and seconds. Compare supposes that months value has more priority than days, and days has more priority than seconds.
Syntax

```pascal
procedure DecodeInterval(var Years: integer; var Months: integer; var Days: integer; var Hours: integer; var Minutes: integer; var Seconds: integer; var Microseconds: integer);
```

**Parameters**

- **Years**
  - Holds the years value.
- **Months**
  - Holds the months value.
- **Days**
  - Holds the days value.
- **Hours**
  - Holds the hours value.
- **Minutes**
  - Holds the minutes value.
- **Seconds**
  - Holds the seconds value.
- **Microseconds**
  - Holds the microseconds value.

**Remarks**

Call the DecodeInterval method to get the value of interval as years, months, days, hours, minutes, seconds, and microseconds.

© 1997-2021 Devart. All Rights Reserved.

5.22.1.7.3.4 EncodeInterval Method

Sets the value of interval as years, months, days, hours, minutes, seconds, and microseconds.

**Class**

TPgInterval

Syntax

```pascal
procedure EncodeInterval(Years: integer; Months: integer; Days: integer; Hours: integer; Minutes: integer; Seconds: integer;
```
Microseconds: integer);

**Parameters**

*Years*
  Holds the years value.

*Months*
  Holds the months value.

*Days*
  Holds the days value.

*Hours*
  Holds the hours value.

*Minutes*
  Holds the minutes value.

*Seconds*
  Holds the seconds value.

*Microseconds*
  Holds the microseconds value.

**Remarks**

Call the EncodeInterval method to set the value of interval as years, months, days, hours, minutes, seconds, and microseconds.

© 1997-2021 Devart. All Rights Reserved.

5.22.1.8 **TPgLSeg Class**

A class for working with PostgreSQL LSEG (line segment) data type.

For a list of all members of this type, see [TPgLSeg](#) members.

**Unit**

*PgObjects*

**Syntax**

```
TPgLSeg = class(TPgPointsArray);
```

**Remarks**

The TPgLSeg class is used to work with PostgreSQL LSEG (line segment) data type.
Inheritance Hierarchy

TSharedObject
TPgGeometric
  TPgPointsArray
  TPgLSeg

© 1997-2021 Devart. All Rights Reserved.

5.22.1.8.1 Members

**TPgLSeg** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td>EndPoint</td>
<td>Holds the end point of the line segment.</td>
</tr>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>StartPoint</td>
<td>Holds the start point of the line segment.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Assigns a value from another TPgGeometric object to this object.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
5.22.1.8.2 Properties

Properties of the `TPgLSeg` class.

For a complete list of the `TPgLSeg` class members, see the [TPgLSeg Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AsString</code> (inherited from <code>TPgGeometric</code>)</td>
<td>Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td><code>EndPoint</code></td>
<td>Holds the end point of the line segment.</td>
</tr>
<tr>
<td><code>RefCount</code> (inherited from <code>TSharedObject</code>)</td>
<td>Used to return the count of reference to a <code>TSharedObject</code> object.</td>
</tr>
<tr>
<td><code>StartPoint</code></td>
<td>Holds the start point of the line segment.</td>
</tr>
</tbody>
</table>

### See Also
- [TPgLSeg Class](#)
- [TPgLSeg Class Members](#)

© 1997-2021 Devart. All Rights Reserved.

5.22.1.8.2.1 EndPoint Property

Holds the end point of the line segment.

### Class

`TPgLSeg`

### Syntax

```
property EndPoint: TPgPoint;
```

### Remarks

The EndPoint property holds the end point of the line segment.

© 1997-2021 Devart. All Rights Reserved.
5.22.1.8.2.2 StartPoint Property

Holds the start point of the line segment.

Class

TPgLSeg

Syntax

```property StartPoint: TPgPoint;```

Remarks

The StartPoint property holds the start point of the line segment.

© 1997-2021
Devart. All Rights Reserved.

5.22.1.9 TPgPath Class

A class for working with PostgreSQL PATH data type.

For a list of all members of this type, see TPgPath members.

Unit

PgObjects

Syntax

```TPgPath = class(TPgPointsArray);```

Remarks

The TPgPath class is used to work with PostgreSQL PATH data type.

Inheritance Hierarchy

TSharedObject
  TPgGeometric
  TPgPointsArray
  TPgPath

© 1997-2021
Devart. All Rights Reserved.
5.22.1.9.1 Members

**TPgPath** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>(inherited from <strong>TPgGeometric</strong>) Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td>Count</td>
<td>Holds the count of points in the path.</td>
</tr>
<tr>
<td>IsClosedPath</td>
<td>If True, the path is closed.</td>
</tr>
<tr>
<td>Points</td>
<td>Used to access a point in the path by its index.</td>
</tr>
<tr>
<td>RefCount</td>
<td>(inherited from <strong>TSharedObject</strong>) Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>(inherited from <strong>TSharedObject</strong>) Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>(inherited from <strong>TPgGeometric</strong>) Assigns a value from another TPgGeometric object to this object.</td>
</tr>
<tr>
<td>Release</td>
<td>(inherited from <strong>TSharedObject</strong>) Decrements the reference count.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.22.1.9.2 Properties

Properties of the **TPgPath** class.

For a complete list of the **TPgPath** class members, see the [TPgPath Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 2021 Devart
AsString (inherited from TPgGeometric)

Used to get and set the geometric value as string.

Count

Holds the count of points in the path.

IsClosedPath

If True, the path is closed.

Points

Used to access a point in the path by its index.

RefCount (inherited from TSharedObject)

Used to return the count of reference to a TSharedObject object.

See Also

- TPgPath Class
- TPgPath Class Members

Class

TPgPath

Syntax

property Count: integer;

Remarks

The Count property holds the count of points in the path.
Property `IsClosedPath`: boolean;

Remarks

If the `IsClosedPath` property is True, the path is closed. This assumes that the last point in the path is connected with the first point by a line.

If False, the path is not closed.

Class `TPgPath`

Syntax

```property Points: array of TPgPoint;```

Remarks

Use the Points property to access a point in the path by its index.

5.22.1.10 `TPgPoint` Class

A class for working with PostgreSQL POINT data type.

For a list of all members of this type, see `TPgPoint` members.

Unit `PgObjects`

Syntax

```TPgPoint = class(TPgGeometric);```
Remarks
The TPgPoint class is used to work with PostgreSQL POINT data type.

Inheritance Hierarchy
TPgPoint
© 1997-2021 Devart. All Rights Reserved.

5.22.1.10.1 Members

**TPgPoint** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString (inherited from <strong>TPgGeometric</strong>)</td>
<td>Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td>RefCount (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>X</td>
<td>Holds the X coordinate of the point.</td>
</tr>
<tr>
<td>Y</td>
<td>Holds the Y coordinate of the point.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef (inherited from <strong>TSharedObject</strong>)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Assigns a value from another TPgPoint object to this object.</td>
</tr>
<tr>
<td>Release (inherited from <strong>TSharedObject</strong>)</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>
5.22.1.10.2 Properties

Properties of the **TPgPoint** class.

For a complete list of the **TPgPoint** class members, see the [TPgPoint Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsString</strong> (inherited from <strong>TPgGeometric</strong>)</td>
<td>Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td><strong>RefCount</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td><strong>X</strong></td>
<td>Holds the X coordinate of the point.</td>
</tr>
<tr>
<td><strong>Y</strong></td>
<td>Holds the Y coordinate of the point.</td>
</tr>
</tbody>
</table>

**See Also**
- [TPgPoint Class](#)
- [TPgPoint Class Members](#)
5.22.1.10.2.2 Y Property

Holds the Y coordinate of the point.

Class
TPgPoint

Syntax

```plaintext
property Y: Double;
```

Remarks

The Y property holds the Y coordinate of the point.

5.22.1.10.3 Methods

Methods of the TPgPoint class.

For a complete list of the TPgPoint class members, see the TPgPoint Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Assigns a value from another TPgPoint object to this object.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

See Also

- TPgPoint Class
- TPgPoint Class Members
5.22.1.10.3.1 Assign Method

Assigns a value from another TPgPoint object to this object.

Class
TPgPoint

Syntax

```plaintext
procedure Assign(Source: TPgGeometric); override;
```

Parameters

- **Source**
  Holds the TPgPoint object to assign value from.

Remarks

Call the Assign method to assign a value from another TPgPoint object to this object.

5.22.1.11 TPgPointsArray Class

A base class for working with geometric data types.

For a list of all members of this type, see TPgPointsArray members.

Unit
PgObjects

Syntax

```plaintext
TPgPointsArray = class(TPgGeometric);
```

Remarks

The TPgPointsArray is a base class for most classes that work with geometric data types.

Inheritance Hierarchy
**TPgPointsArray** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Assigns a value from another TPgGeometric object to this object.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

**5.22.1.12 TPgPolygon Class**

A class for working with PostgreSQL POLYGON data type.

For a list of all members of this type, see **TPgPolygon** members.
**PgObjects**

Syntax

```cpp
TPgPolygon = class(TPgPointsArray);
```

Remarks

The `TPgPolygon` class is used to work with PostgreSQL POLYGON data type.

Inheritance Hierarchy

```
TSharedObject
   TPgGeometric
      TPgPointsArray
         TPgPolygon
```

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AsString</code></td>
<td>(inherited from <code>TPgGeometric</code>) Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td><code>Count</code></td>
<td>Holds the count of points in the polygon.</td>
</tr>
<tr>
<td><code>Points</code></td>
<td>Used to access a point in the polygon by its index.</td>
</tr>
<tr>
<td><code>RefCount</code></td>
<td>(inherited from <code>TSharedObject</code>) Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AddRef</code></td>
<td>(inherited from <code>TSharedObject</code>) Increments the reference count for the number of references dependent on the</td>
</tr>
</tbody>
</table>
Properties of the **TPgPolygon** class.

For a complete list of the **TPgPolygon** class members, see the [TPgPolygon Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AsString</strong> (inherited from <strong>TPgGeometric</strong>)</td>
<td>Used to get and set the geometric value as string.</td>
</tr>
<tr>
<td><strong>Count</strong></td>
<td>Holds the count of points in the polygon.</td>
</tr>
<tr>
<td><strong>Points</strong></td>
<td>Used to access a point in the polygon by its index.</td>
</tr>
<tr>
<td><strong>RefCount</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
</tbody>
</table>

**See Also**
- [TPgPolygon Class](#)
- [TPgPolygon Class Members](#)

### 5.22.1.12.2.1 Count Property

Holds the count of points in the polygon.
## Points Property

### Syntax

```plaintext
property Points: array of TPgPoint;
```

### Remarks

Use the Points property to access a point in the polygon by its index.

## TPgRefCursor Class

### Syntax

```plaintext
TPgRefCursor = class(TPgCursor);
```

### Remarks

A class for working with REFCURSOR values.

For a list of all members of this type, see [TPgRefCursor](#) members.
The TPgRefCursor class is used to work with REFCURSOR values. An instance of TPgRefCursor holds the cursor name that is returned, for example, from a stored procedure, and can be used to fetch data from the cursor. Assign an instance of the TPgRefCursor class to the Cursor property of TPgQuery and call the TPgQuery.Open method to fetch data from the cursor.

**Note:** all operations with REFCURSOR require an active transaction.

### Inheritance Hierarchy

```
TSharedObject
   TCRCursor
   TPgCursor
   TPgRefCursor
```

5.22.1.13.1 Members

**TPgRefCursor** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CursorName</strong></td>
<td>Used to get the name of a cursor.</td>
</tr>
<tr>
<td><strong>RefCount</strong> (inherited from TSharedObject)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td><strong>State</strong> (inherited from TPgCursor)</td>
<td>Used to set the cursor state.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddRef</strong> (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td><strong>Release</strong> (inherited from TSharedObject)</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>
Properties of the **TPgRefCursor** class.

For a complete list of the **TPgRefCursor** class members, see the [TPgRefCursor Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CursorName</td>
<td>Used to get the name of a cursor.</td>
</tr>
<tr>
<td>RefCount (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>State (inherited from <strong>TPgCursor</strong>)</td>
<td>Used to set the cursor state.</td>
</tr>
</tbody>
</table>

See Also
- [TPgRefCursor Class](#)
- [TPgRefCursor Class Members](#)

© 1997-2021 Devart. All Rights Reserved.

5.22.1.13.2.1 CursorName Property

Used to get the name of a cursor.

**Class**

**TPgRefCursor**

**Syntax**

```property CursorName: string;```

**Remarks**

Use the CursorName property to get the name of a cursor.
5.22.1.14 TPgRow Class

A class for working with PostgreSQL composite (ROW) data types.
For a list of all members of this type, see TPgRow members.

Unit
PgObjects

Syntax

TPgRow = class(TDBObject);

Remarks
The TPgRow class is used to work with PostgreSQL composite (ROW) data types.

Inheritance Hierarchy

TSharedObject
  TDBObject
  TPgRow

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to get and set the row value as string.</td>
</tr>
<tr>
<td>AttrAsPgBox</td>
<td>Used to get a reference to a TPgBox object for an attribute of ftPgBox type.</td>
</tr>
<tr>
<td>AttrAsPgCircle</td>
<td>Used to get a reference to a TPgCircle object for an attribute of ftPgCircle type.</td>
</tr>
<tr>
<td>AttrAsPgCursor</td>
<td>Used to get a reference to a</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AttrAsPgDate</td>
<td>Used to get a reference to TPgDate object for attribute of ftPgDate type.</td>
</tr>
<tr>
<td>AttrAsPgInterval</td>
<td>Used to get a reference to a TPgInterval object for an attribute of ftPgInterval type.</td>
</tr>
<tr>
<td>AttrAsPgLargeObject</td>
<td>Used to get a reference to a TPgLargeObject object for an attribute of ftPgLargeObject type.</td>
</tr>
<tr>
<td>AttrAsPgLSeg</td>
<td>Used to get a reference to a TPgLSeg object for an attribute of ftPgLSeg type.</td>
</tr>
<tr>
<td>AttrAsPgPath</td>
<td>Used to get a reference to a TPgPath object for an attribute of ftPgPath type.</td>
</tr>
<tr>
<td>AttrAsPgPoint</td>
<td>Used to get a reference to a TPgPoint object for an attribute of ftPgPoint type.</td>
</tr>
<tr>
<td>AttrAsPgPolygon</td>
<td>Used to get a reference to a TPgPolygon object for an attribute of ftPgPolygon type.</td>
</tr>
<tr>
<td>AttrAsPgRow</td>
<td>Used to get a reference to a TPgRow object for an attribute of composite type.</td>
</tr>
<tr>
<td>AttrAsPgTime</td>
<td>Used to get a reference to a TPgTime object for an attribute of ftPgTime type.</td>
</tr>
<tr>
<td>AttrAsPgTimeStamp</td>
<td>Used to get a reference to a TPgTimeStamp object for an attribute of ftPgTimeStamp type.</td>
</tr>
<tr>
<td>AttrIsNull</td>
<td>Used to find out if an attribute is NULL, or to set an attribute value to NULL.</td>
</tr>
<tr>
<td>AttrValue</td>
<td>Used to get or set the value of an attribute.</td>
</tr>
<tr>
<td>RefCount (inherited from TSharedObject)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>RowType</td>
<td>Holds the reference to a TPgRowType object</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Assigns a value from another TPgRow object to this object.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

Properties of the TPgRow class.

For a complete list of the TPgRow class members, see the TPgRow Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>Used to get and set the row value as string.</td>
</tr>
<tr>
<td>AttrAsPgBox</td>
<td>Used to get a reference to a TPgBox object for an attribute of ftPgBox type.</td>
</tr>
<tr>
<td>AttrAsPgCircle</td>
<td>Used to get a reference to a TPgCircle object for an attribute of ftPgCircle type.</td>
</tr>
<tr>
<td>AttrAsPgCursor</td>
<td>Used to get a reference to a TPgCursor object for an attribute of ftPgCursor type.</td>
</tr>
<tr>
<td>AttrAsPgDate</td>
<td>Used to get a reference to TPgDate object for attribute of ftPgDate type.</td>
</tr>
</tbody>
</table>
### AttrAsPgInterval
Used to get a reference to a TPgInterval object for an attribute of ftPgInterval type.

### AttrAsPgLargeObject
Used to get a reference to a TPgLargeObject object for an attribute of ftPgLargeObject type.

### AttrAsPgLSeg
Used to get a reference to a TPgLSeg object for an attribute of ftPgLSeg type.

### AttrAsPgPath
Used to get a reference to a TPgPath object for an attribute of ftPgPath type.

### AttrAsPgPoint
Used to get a reference to a TPgPoint object for an attribute of ftPgPoint type.

### AttrAsPgPolygon
Used to get a reference to a TPgPolygon object for an attribute of ftPgPolygon type.

### AttrAsPgRow
Used to get a reference to a TPgRow object for an attribute of composite type.

### AttrAsPgTime
Used to get a reference to a TPgTime object for an attribute of ftPgTime type.

### AttrAsPgTimeStamp
Used to get a reference to a TPgTimeStamp object for an attribute of ftPgTimeStamp type.

### AttrIsNull
Used to find out if an attribute is NULL, or to set an attribute value to NULL.

### AttrValue
Used to get or set the value of an attribute.

### RefCount (inherited from TSharedObject)
Used to return the count of reference to a TSharedObject object.

### RowType
Holds the reference to a TPgRowType object containing information about corresponding composite type.

### See Also
5.22.1.14.2.1  AsString Property

Used to get and set the row value as string.

Class

TPgRow

Syntax

```property AsString: string;```

Remarks

Use the AsString property to get and set the row value as string.

5.22.1.14.2.2  AttrAsPgBox Property(Indexer)

Used to get a reference to a TPgBox object for an attribute of ftPgBox type.

Class

TPgRow

Syntax

```property AttrAsPgBox[const Name: string]: TPgBox;```

Parameters

- **Name**
  - Holds the name of an attribute.

Remarks

Use the AttrAsPgBox property to get a reference to a TPgBox object for an attribute of ftPgBox type.
5.22.1.14.2.3 AttrAsPgCircle Property (Indexer)

Used to get a reference to a TPgCircle object for an attribute of ftPgCircle type.

Class
TPgRow

Syntax

```
property AttrAsPgCircle[const Name: string]: TPgCircle;
```

Parameters

Name
Holds the name of an attribute.

Remarks

Use the AttrAsPgCircle property to get a reference to a TPgCircle object for an attribute of ftPgCircle type.

5.22.1.14.2.4 AttrAsPgCursor Property (Indexer)

Used to get a reference to a TPgCursor object for an attribute of ftPgCursor type.

Class
TPgRow

Syntax

```
property AttrAsPgCursor[const Name: string]: TPgCursor;
```

Parameters

Name
Holds the name of an attribute.

Remarks

Use the AttrAsPgCursor property to get a reference to a TPgCursor object for an attribute of ftPgCursor type.
5.22.1.14.2.5 AttrAsPgDate Property (Indexer)

Used to get a reference to TPgDate object for attribute of ftPgDate type.

Class
TPgRow

Syntax

```property AttrAsPgDate[const Name: string]: TPgDate;```

Parameters

Name
Holds the name of an attribute.

Remarks

Use the AttrAsPgDate property to get a reference to a TPgDate object for an attribute of ftPgDate type.

5.22.1.14.2.6 AttrAsPgInterval Property (Indexer)

Used to get a reference to TPgInterval object for an attribute of ftPgInterval type.

Class
TPgRow

Syntax

```property AttrAsPgInterval[const Name: string]: TPgInterval;```

Parameters

Name
Holds the name of an attribute.

Remarks
Use the AttrAsPgInterval property to get a reference to a TPgInterval object for an attribute of ftPgInterval type.

Class
- **TPgRow**

Syntax
```pascal
property AttrAsPgInterval[const Name: string]: TPgInterval;
```

Parameters
- **Name**
  - Holds the name of an attribute.

Remarks
Use the AttrAsPgInterval property to get a reference to a TPgInterval object for an attribute of ftPgInterval type.

Used to get a reference to a TPgLargeObject object for an attribute of ftPgLargeObject type.

Class
- **TPgRow**

Syntax
```pascal
property AttrAsPgLargeObject[const Name: string]: TPgSQLLargeObject;
```

Parameters
- **Name**
  - Holds the name of an attribute.

Remarks
Use the AttrAsPgLargeObject property to get a reference to a TPgLargeObject object for an attribute of ftPgLargeObject type.

Used to get a reference to a TPgLSeg object for an attribute of ftPgLSeg type.

Class
- **TPgRow**

Syntax
```pascal
property AttrAsPgLSeg[const Name: string]: TPgLSeg;
```

Parameters
- **Name**
Holds the name of an attribute.

Remarks
Use the AttrAsPgLSeg property to get a reference to a TPgLSeg object for an attribute of ftPgLSeg type.

Class
TPgRow

Syntax

```pascal
property AttrAsPgLSeg[const Name: string]: TPgLSeg;
```

Parameters
Name
Holds the name of an attribute.

Remarks
Use the AttrAsPgLSeg property to get a reference to a TPgLSeg object for an attribute of ftPgLSeg type.

Class
TPgRow

Syntax

```pascal
property AttrAsPgPath[const Name: string]: TPgPath;
```

Parameters
Name
Holds the name of an attribute.

Remarks
Use the AttrAsPgPath property to get a reference to a TPgPath object for an attribute of ftPgPath type.

Class
TPgRow

Syntax

```pascal
property AttrAsPgPoint[const Name: string]: TPgPoint;
```

Parameters
Name
Holds the name of an attribute.

Remarks
Use the AttrAsPgPoint property to get a reference to a TPgPoint object for an attribute of ftPgPoint type.

Class
TPgRow

Syntax

```
property AttrAsPgPolygon[const Name: string]: TPgPolygon;
```

Parameters

Name
Holds the name of an attribute.

Remarks
Use the AttrAsPgPolygon property to get a reference to a TPgPolygon object for an attribute of ftPgPolygon type.

Class
TPgRow

Syntax

```
property AttrAsPgRow[const Name: string]: TPgRow;
```
Parameters

Name
Holds the name of an attribute.

Remarks
Use the AttrAsPgRow property to get a reference to a TPgRow object for an attribute of composite type.

Class
TPgRow

Syntax

```delphi
property AttrAsPgRow[const Name: string]: TPgRow;
```

Parameters

Name
Holds the name of an attribute.

Remarks
Use the AttrAsPgRow property to get a reference to a TPgRow object for an attribute of ftPgTime type.

Class
TPgRow

Syntax

```delphi
property AttrAsPgTime[const Name: string]: TPgTime;
```

Parameters

Name
Holds the name of an attribute.

Remarks
Use the AttrAsPgTime property to get a reference to a TPgTime object for an attribute of ftPgTime type.

Class
TPgRow

Syntax

```delphi
property AttrAsPgTimeStamp[const Name: string]: TPgTimeStamp;
```

Parameters

Name
Holds the name of an attribute.

Remarks
Use the AttrAsPgTimeStamp property to get a reference to a TPgTimeStamp object for an attribute of ftPgTimeStamp type.
**property** AttrAsPgTimeStamp[const Name: string]: TPgTimeStamp;

**Parameters**

Name
Holds the name of an attribute.

**Remarks**

Use the AttrAsPgTimeStamp property to get a reference to a TPgTimeStamp object for an attribute of ftPgTimeStamp type.

© 1997-2021 Devart. All Rights Reserved.

5.22.1.14.2.15 AttrIsNull Property(Indexer)

Used to find out if an attribute is NULL, or to set an attribute value to NULL.

**Class**
TPgRow

**Syntax**

```
property AttrIsNull[const Name: string]: boolean;
```

**Parameters**

Name
Holds the name of an attribute.

**Remarks**

Use the AttrIsNull property to find out if an attribute is NULL, or to set an attribute value to NULL.

© 1997-2021 Devart. All Rights Reserved.

5.22.1.14.2.16 AttrValue Property(Indexer)

Used to get or set the value of an attribute.

**Class**
TPgRow
Syntax

```pascal
property AttrValue[const Name: string]: variant;
```

Parameters

Name
Holds the name of an attribute.

Remarks

Use the AttrValue property to get or set the value of an attribute.

Class

TPgRow

Syntax

```pascal
property RowType: TPgRowType;
```

Remarks

The RowType property holds the reference to a TPgRowType object containing information about corresponding composite type. This property must be set before you can read or write values to attributes.
### Assign Method

Assigns a value from another TPgRow object to this object.

#### Class

**TPgRow**

#### Syntax

```plaintext
procedure Assign(Source: TPgRow);
```

#### Parameters

- **Source**
  - Holds the TPgRow object to assign a value from.

#### Remarks

Call the `Assign` method to assign a value from another TPgRow object to this object.
TPgRowType Class

A class holding the description of a PostgreSQL composite (ROW) type.

For a list of all members of this type, see TPgRowType members.

Unit

PgObjects

Syntax

TPgRowType = class(TPgType);

Remarks

The TPgRowType class holds the description of a PostgreSQL composite (ROW) type. Use the GetRowType method of TPgConnection to get an instance of a TPgRowType object.

Inheritance Hierarchy

TSharedObject
  TObjectType
    TPgType
      TPgRowType

Members

TPgRowType class overview.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describe</td>
<td>Overloaded. Provides information about a composite type from the database.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
5.22.1.15.2.1 Describe Method

Provides information about a composite type from the database.

Class
TPgRowType

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describe</td>
<td>Provides information about a composite type from the database by the OID.</td>
</tr>
<tr>
<td>Describe</td>
<td>Provides information about a composite type from the database by the name.</td>
</tr>
</tbody>
</table>

Provides information about a composite type from the database by the OID.

Unit

Syntax

Remarks
Call the Describe method to get information about a composite type from the database by the OID.
Provides information about a composite type from the database by the name.

Unit

Syntax

Remarks

Call the Describe method to get information about a composite type from the database by the name.

5.22.1.16 TPgSQLLargeObject Class

A class for working with PostgreSQL large objects.

For a list of all members of this type, see TPgSQLLargeObject members.

Unit

PgObjects

Syntax

TPgSQLLargeObject = class(TCompressedBlob);

Remarks

The TPgSQLLargeObject class is used to work with PostgreSQL large objects. An instance of TPgSQLLargeObject holds an OID of large object and can be used for operations with this large object.

Note: all operations with large objects require an active transaction.

Inheritance Hierarchy

TSharedObject
  TBlob
    TCompressedBlob
**TPgSQLLargeObject** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString (inherited from <strong>TBlob</strong>)</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td>AsWideString (inherited from <strong>TBlob</strong>)</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>Cached</td>
<td>Used to specify whether the content of large object should be stored in memory buffer on the client.</td>
</tr>
<tr>
<td>Compressed (inherited from <strong>TCompressedBlob</strong>)</td>
<td>Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td>CompressedSize (inherited from <strong>TCompressedBlob</strong>)</td>
<td>Used to indicate compressed size of the Blob data.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object that is used to perform operations with large object.</td>
</tr>
<tr>
<td>IsUnicode (inherited from <strong>TBlob</strong>)</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td>OID</td>
<td>Holds the OID of a large object.</td>
</tr>
<tr>
<td>RefCount (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size (inherited from <strong>TBlob</strong>)</td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

---

© 2021 Devart
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddRef</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td><strong>Assign</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td><strong>Clear</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td><strong>CloseObject</strong></td>
<td>Closes the large object that was previously opened by the OpenObject method.</td>
</tr>
<tr>
<td><strong>CreateObject</strong></td>
<td>Creates a new large object database.</td>
</tr>
<tr>
<td><strong>LoadFromFile</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Loads the contents of a file into a TBlob object.</td>
</tr>
<tr>
<td><strong>LoadFromStream</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Copies the contents of a stream into the TBlob object.</td>
</tr>
<tr>
<td><strong>OpenObject</strong></td>
<td>Opens the large object specified by the OID property.</td>
</tr>
<tr>
<td><strong>Read</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlob.</td>
</tr>
<tr>
<td><strong>ReadBlob</strong></td>
<td>Reads the content of the large object from the database.</td>
</tr>
<tr>
<td><strong>Release</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Decrements the reference count.</td>
</tr>
<tr>
<td><strong>SaveToFile</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Saves the contents of the TBlob object to a file.</td>
</tr>
<tr>
<td><strong>SaveToStream</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Copies the contents of a TBlob object to a stream.</td>
</tr>
<tr>
<td><strong>Truncate</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Sets new TBlob size and discards all data over it.</td>
</tr>
<tr>
<td><strong>UnlinkObject</strong></td>
<td>Deletes the large object specified by the OID property from the database.</td>
</tr>
<tr>
<td><strong>Write</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Stores a raw sequence of bytes into a TBlob object.</td>
</tr>
<tr>
<td><strong>WriteBlob</strong></td>
<td>Writes the content of the large object to the database.</td>
</tr>
</tbody>
</table>
Properties of the TPgSQLLargeObject class.

For a complete list of the TPgSQLLargeObject class members, see the TPgSQLLargeObject Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString (inherited from TBlob)</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td>AsWideString (inherited from TBlob)</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td>Cached</td>
<td>Used to specify whether the content of large object should be stored in memory buffer on the client.</td>
</tr>
<tr>
<td>Compressed (inherited from TCompressedBlob)</td>
<td>Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td>CompressedSize (inherited from TCompressedBlob)</td>
<td>Used to indicate compressed size of the Blob data.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object that is used to perform operations with large object.</td>
</tr>
<tr>
<td>IsUnicode (inherited from TBlob)</td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td>OID</td>
<td>Holds the OID of a large object.</td>
</tr>
<tr>
<td>RefCount (inherited from TSharedObject)</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Size (inherited from TBlob)</td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
</tbody>
</table>

See Also
- TPgSQLLargeObject Class
- TPgSQLLargeObject Class Members
5.22.1.16.2.1 Cached Property

Used to specify whether the content of large object should be stored in memory buffer on the client.

**Class**  
TPgSQLLargeObject

**Syntax**

```
property Cached: boolean;
```

**Remarks**

Use the Cached property to specify whether the content of large object should be stored in memory buffer on the client.

By default the whole content of large object is stored in the memory buffer on the client. Set Cached to False if you want to reduce memory usage in your application when working with very large object. When Cached is False, the Read and Write methods read/write data directly from/to the database.

© 1997-2021  
Devart. All Rights Reserved.

5.22.1.16.2.2 Connection Property

Used to specify a connection object that is used to perform operations with large object.

**Class**  
TPgSQLLargeObject

**Syntax**

```
property Connection: TPgSQLConnection;
```

**Remarks**

Use the Connection property to specify a connection object that is used to perform operations with large object.
5.22.1.16.2.3 OID Property

Holds the OID of a large object.

Class

**TPgSQLLargeObject**

Syntax

```property
OID: OID;
```

Remarks

Read the OID property to get the OID of the object created by the CreateObject method. Set the OID property to work with an existing large object.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddRef</strong></td>
<td>(inherited from <strong>TSharedObject</strong>) Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td><strong>Assign</strong></td>
<td>(inherited from <strong>TBlob</strong>) Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td><strong>Clear</strong></td>
<td>(inherited from <strong>TBlob</strong>) Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td><strong>CloseObject</strong></td>
<td>Closes the large object that was previously opened by the OpenObject method.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CreateObject</td>
<td>Creates a new large object database.</td>
</tr>
<tr>
<td>LoadFromFile (inherited from TBlob)</td>
<td>Loads the contents of a file into a TBlob object.</td>
</tr>
<tr>
<td>LoadFromStream (inherited from TBlob)</td>
<td>Copies the contents of a stream into the TBlob object.</td>
</tr>
<tr>
<td>OpenObject</td>
<td>Opens the large object specified by the OID property.</td>
</tr>
<tr>
<td>Read (inherited from TBlob)</td>
<td>Acquires a raw sequence of bytes from the data stored in TBlob.</td>
</tr>
<tr>
<td>ReadBlob</td>
<td>Reads the content of the large object from the database.</td>
</tr>
<tr>
<td>Release (inherited from TSharedObject)</td>
<td>Decrement the reference count.</td>
</tr>
<tr>
<td>SaveToFile (inherited from TBlob)</td>
<td>Saves the contents of the TBlob object to a file.</td>
</tr>
<tr>
<td>SaveToStream (inherited from TBlob)</td>
<td>Copies the contents of a TBlob object to a stream.</td>
</tr>
<tr>
<td>Truncate (inherited from TBlob)</td>
<td>Sets new TBlob size and discards all data over it.</td>
</tr>
<tr>
<td>UnlinkObject</td>
<td>Deletes the large object specified by the OID property from the database.</td>
</tr>
<tr>
<td>Write (inherited from TBlob)</td>
<td>Stores a raw sequence of bytes into a TBlob object.</td>
</tr>
<tr>
<td>WriteBlob</td>
<td>Writes the content of the large object to the database.</td>
</tr>
</tbody>
</table>

See Also

- TPgSQLLargeObject Class
- TPgSQLLargeObject Class Members

© 1997-2021
Devart. All Rights Reserved.

5.22.1.16.3.1  CloseObject Method

Closes the large object that was previously opened by the OpenObject method.

Class
**TPgSQLLargeObject**

**Syntax**

```plaintext
procedure CloseObject;
```

**Remarks**

Call the `CloseObject` property to close the large object that was previously opened by the `OpenObject` method.

© 1997-2021 Devart. All Rights Reserved.

---

### 5.22.1.16.3.2 CreateObject Method

Creates a new large object database.

**Class**

TPgSQLLargeObject

**Syntax**

```plaintext
procedure CreateObject;
```

**Remarks**

Call the `CreateObject` method to create a new large object database.

© 1997-2021 Devart. All Rights Reserved.

---

### 5.22.1.16.3.3 OpenObject Method

Opens the large object specified by the OID property.

**Class**

TPgSQLLargeObject

**Syntax**

```plaintext
procedure OpenObject;
```

**Remarks**
5.22.1.16.3.4 ReadBlob Method

Reads the content of the large object from the database.

Class

TPgSQLLargeObject

Syntax

```
procedure ReadBlob;
overload;
procedure ReadBlob(var SharedPiece: PPieceHeader);
overload;
```

Remarks

Call the ReadBlob method to get the content of large object from the database. When reading such properties as AsString or AsWideString, this method is called automatically.

5.22.1.16.3.5 UnlinkObject Method

Deletes the large object specified by the OID property from the database.

Class

TPgSQLLargeObject

Syntax

```
procedure UnlinkObject;
```

Remarks

Call the UnlinkObject method to delete the large object specified by the OID property from the database.
5.22.1.16.3.6 WriteBlob Method

Writes the content of the large object to the database.

Class

TPgSQLLargeObject

Syntax

```
procedure writeBlob;
```

Remarks

Call the WriteBlob method to write the content of the large object to the database.

When the Cached property is set to True, call WriteLob to write the cached content of the large object to the database.

5.22.1.17 TPgTime Class

A class for working with PostgreSQL TIME and TIME WITH TIMEZONE data type.

For a list of all members of this type, see TPgTime members.

Unit

PgObjects

Syntax

```
TPgTime = class(TCustomPgTimeStamp);
```

Remarks

The TPgTime class is used to work with PostgreSQL TIME and TIME WITH TIMEZONE data type.

Inheritance Hierarchy

TSharedObject

  TCustomPgTimeStamp

  TPgTime
**TPgTime** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsDateTime</td>
<td>Used to get and set the timestamp value as TDateTime.</td>
</tr>
<tr>
<td>AsSQLTimeStamp</td>
<td>Used to get and set the timestamp value as TSQLTimeStamp record.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to get and set the timestamp value as string.</td>
</tr>
<tr>
<td>Days</td>
<td>Holds the date part of timestamp.</td>
</tr>
<tr>
<td>HasTimeZone</td>
<td>Used to specify whether the timestamp object has time zone.</td>
</tr>
<tr>
<td>IsInfinity</td>
<td>Determines if a timestamp object has +/-infinity value.</td>
</tr>
<tr>
<td>IsNegInfinity</td>
<td>Determines if a timestamp object has -infinity value.</td>
</tr>
<tr>
<td>IsPosInfinity</td>
<td>Determines if a timestamp object has +infinity value.</td>
</tr>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Ticks</td>
<td>Holds the time part of timestamp in microseconds.</td>
</tr>
<tr>
<td>TimeZoneOffset</td>
<td>Used to get or set the time zone offset.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the</td>
</tr>
</tbody>
</table>
### Properties of the `TPgTime` class.

For a complete list of the `TPgTime` class members, see the [TPgTime Members](#) topic.

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsDateTime (inherited from <code>TCustomPgTimeStamp</code>)</td>
<td>Used to get and set the timestamp value as</td>
</tr>
</tbody>
</table>
### TimeZoneOffset property

**Usage**

- **Type:** integer

Used to get or set the time zone offset.

**Syntax**

```pascal
property TimeZoneOffset: integer;
```
Remarks

Use the TimeZoneOffset property to get or set the time zone offset (in seconds).

For values of TIME WITH TIMEZONE use this property to get or set offset of time value from Universal Time (UTC).

Value of TimeZoneOffset is in seconds.

Syntax

```cpp
TPgTimeStamp = class(TCustomPgTimeStamp);
```

Remarks

The TPgTimeStamp class is used to work with PostgreSQL TIMESTAMP and TIMESTAMP WITH TIMEZONE data types.

Inheritance Hierarchy

```
TSharedObject
  TCustomPgTimeStamp
    TPgTimeStamp
```

5.22.1.18.1 Members

**TPgTimeStamp** class overview.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsDateTime</td>
<td>Used to get and set the timestamp value as TDateTime.</td>
</tr>
<tr>
<td>AsSQLTime Stamp</td>
<td>Used to get and set the timestamp value as TSQLTimeStamp record.</td>
</tr>
<tr>
<td>AsString</td>
<td>Used to get and set the timestamp value as string.</td>
</tr>
<tr>
<td>Days</td>
<td>Holds the date part of timestamp.</td>
</tr>
<tr>
<td>HasTimeZone</td>
<td>Used to specify whether the timestamp object has time zone.</td>
</tr>
<tr>
<td>IsInfinity</td>
<td>Determines if a timestamp object has +/-infinity value.</td>
</tr>
<tr>
<td>IsNegInfinity</td>
<td>Determines if a timestamp object has -infinity value.</td>
</tr>
<tr>
<td>IsPosInfinity</td>
<td>Determines if a timestamp object has +infinity value.</td>
</tr>
<tr>
<td>RefCount</td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td>Ticks</td>
<td>Holds the time part of timestamp in microseconds.</td>
</tr>
<tr>
<td>TimeZoneOffset</td>
<td>Specifies the time zone offset of a timestamp object (in seconds).</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
<tr>
<td>Assign</td>
<td>Assigns a value from another TCustomPgTimeStamp object to this object.</td>
</tr>
</tbody>
</table>
**Compare** (inherited from **TCustomPgTimeStamp**) | Compares two TCustomPgTimeStamp objects.
---|---
**DecodeDate** (inherited from **TCustomPgTimeStamp**) | Provides year, month, and day from the timestamp object.
**DecodeDateTime** (inherited from **TCustomPgTimeStamp**) | Provides the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
**DecodeTime** (inherited from **TCustomPgTimeStamp**) | Provides hour, minute, second, and microsecond from the timestamp object.
**EncodeDate** (inherited from **TCustomPgTimeStamp**) | Sets year, month, and day in the timestamp object.
**EncodeDateTime** (inherited from **TCustomPgTimeStamp**) | Sets the value of the timestamp object as year, month, day, hour, minute, second, and microsecond.
**EncodeTime** (inherited from **TCustomPgTimeStamp**) | Sets hour, minute, second, and microsecond in the timestamp object.
**Release** (inherited from **TSharedObject**) | Decrements the reference count.

### 5.23 PgScript

This unit contains the implementation of the TPgScript component.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgScript</td>
<td>Executes sequences of SQL statements.</td>
</tr>
</tbody>
</table>
5.23.1 Classes

Classes in the PgScript unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgScript</td>
<td>Executes sequences of SQL statements.</td>
</tr>
</tbody>
</table>

5.23.1.1 TPgScript Class

Executes sequences of SQL statements.

For a list of all members of this type, see TPgScript members.

Unit

PgScript

Syntax

TPgScript = class(TDAScript);

Inheritance Hierarchy

TDAScript
TPgScript

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Used to specify the connection in which the script will be executed.</td>
</tr>
</tbody>
</table>
### DataSet (inherited from TDAStat
tScript)
- **Refers to a dataset that holds the result set of query execution.**

### Debug (inherited from TDAStat
tScript)
- **Used to display the script execution and all its parameter values.**

### Delimiter (inherited from TDAStat
tScript)
- **Used to set the delimiter string that separates script statements.**

### EndLine (inherited from TDAStat
tScript)
- **Used to get the current statement last line number in a script.**

### EndOffset (inherited from TDAStat
tScript)
- **Used to get the offset in the last line of the current statement.**

### EndPos (inherited from TDAStat
tScript)
- **Used to get the end position of the current statement.**

### Macros (inherited from TDAStat
tScript)
- **Used to change SQL script text in design- or run-time easily.**

### SQL (inherited from TDAStat
tScript)
- **Used to get or set script text.**

### StartLine (inherited from TDAStat
tScript)
- **Used to get the current statement start line number in a script.**

### StartOffset (inherited from TDAStat
tScript)
- **Used to get the offset in the first line of the current statement.**

### StartPos (inherited from TDAStat
tScript)
- **Used to get the start position of the current statement in a script.**

### Statements (inherited from TDAStat
tScript)
- **Contains a list of statements obtained from the SQL property.**

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **BreakExec**   | (inherited from TDAStat
tScript)                              | Stops script execution.                                      |
| **ErrorOffset** | (inherited from TDAStat
tScript)                              | Used to get the offset of the statement if the Execute method raised an exception. |
| **Execute**     | (inherited from TDAStat
tScript)                              | Executes a script.                                           |
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteFile (inherited from TDAScript)</td>
<td>Executes SQL statements contained in a file.</td>
</tr>
<tr>
<td>ExecuteNext (inherited from TDAScript)</td>
<td>Executes the next statement in the script and then stops.</td>
</tr>
<tr>
<td>ExecuteStream (inherited from TDAScript)</td>
<td>Executes SQL statements contained in a stream object.</td>
</tr>
<tr>
<td>FindMacro (inherited from TDAScript)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>MacroByName (inherited from TDAScript)</td>
<td>Finds a macro with the specified name.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute (inherited from TDAScript)</td>
<td>Occurs after a SQL script execution.</td>
</tr>
<tr>
<td>BeforeExecute (inherited from TDAScript)</td>
<td>Occurs when taking a specific action before executing the current SQL statement is needed.</td>
</tr>
<tr>
<td>OnError (inherited from TDAScript)</td>
<td>Occurs when PostgreSQL raises an error.</td>
</tr>
</tbody>
</table>

Properties

Properties of the TPgScript class.

For a complete list of the TPgScript class members, see the TPgScript Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSet (inherited from TDAScript)</td>
<td>Refers to a dataset that holds the result set of query execution.</td>
</tr>
<tr>
<td>EndLine (inherited from TDAScript)</td>
<td>Used to get the current statement last line number in a script.</td>
</tr>
<tr>
<td>EndOffset (inherited from TDAScript)</td>
<td>Used to get the offset in the</td>
</tr>
</tbody>
</table>
### TPgScript Class Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Used to specify the connection in which the script will be executed.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the script execution and all its parameter values.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>Used to set the delimiter string that separates script statements.</td>
</tr>
<tr>
<td>Macros</td>
<td>Used to change SQL script text in design- or run-time easily.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to get or set script text.</td>
</tr>
</tbody>
</table>

### Published

#### Name

- **Connection**
- **Debug** (inherited from **TDAScript**)
- **Delimiter** (inherited from **TDAScript**)
- **Macros** (inherited from **TDAScript**)
- **SQL** (inherited from **TDAScript**)

---

**See Also**
- TPgScript Class
- TPgScript Class Members

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
5.23.1.2.1 Connection Property

Used to specify the connection in which the script will be executed.

Class

TPgScript

Syntax

property Connection: TPgConnection;

Remarks

Use the Connection property to specify the connection in which the script will be executed. If a connection has not been opened before running the Execute method, the TPgConnection.Connect method is called.

5.24 PgSQLMonitor

This unit contains the implementation of the TPgSQLMonitor component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPgSQLMonitor</td>
<td>This component serves for monitoring dynamic SQL execution in PgDAC-based applications.</td>
</tr>
</tbody>
</table>

5.24.1 Classes

Classes in the PgSQLMonitor unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
5.24.1.1 TPgSQLMonitor Class

This component serves for monitoring dynamic SQL execution in PgDAC-based applications.

For a list of all members of this type, see TPgSQLMonitor members.

Unit

PgSQLMonitor

Syntax

TPgSQLMonitor = class(TCustomDASQLMonitor);

Remarks

Use TPgSQLMonitor to monitor dynamic SQL execution in PgDAC-based applications. TPgSQLMonitor provides two ways of displaying debug information: with dialog window, DBMonitor or Borland SQL Monitor. Furthermore to receive debug information the TCustomDASQLMonitor.OnSQL event can be used. Also it is possible to use all these ways at the same time, though an application may have only one TPgSQLMonitor object. If an application has no TPgSQLMonitor instance, the Debug window is available to display SQL statements to be sent.

Inheritance Hierarchy

TCustomDASQLMonitor

TPgSQLMonitor

See Also

- TCustomDADataSet.Debug
- TCustomDASQL.Debug
- DBMonitor
5.24.1.1 Members

**TPgSQLMonitor** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong> (inherited from <strong>TCustomDASQLMonitor</strong>)</td>
<td>Used to activate monitoring of SQL.</td>
</tr>
<tr>
<td><strong>DBMonitorOptions</strong> (inherited from <strong>TCustomDASQLMonitor</strong>)</td>
<td>Used to set options for dbMonitor.</td>
</tr>
<tr>
<td><strong>Options</strong> (inherited from <strong>TCustomDASQLMonitor</strong>)</td>
<td>Used to include the desired properties for TCustomDASQLMonitor.</td>
</tr>
<tr>
<td><strong>TraceFlags</strong> (inherited from <strong>TCustomDASQLMonitor</strong>)</td>
<td>Used to specify which database operations the monitor should track in an application at runtime.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnSQL</strong> (inherited from <strong>TCustomDASQLMonitor</strong>)</td>
<td>Occurs when tracing of SQL activity on database components is needed.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

**5.25 VirtualDataSet**

This unit contains implementation of the TVirtualDataSet component.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TCustomVirtualDataSet</strong></td>
<td>A base class for representation of arbitrary data in tabular form.</td>
</tr>
<tr>
<td><strong>TVirtualDataSet</strong></td>
<td>Dataset that processes</td>
</tr>
</tbody>
</table>
Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOnDeleteRecordEvent</td>
<td>This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.</td>
</tr>
<tr>
<td>TOnGetFieldValueEvent</td>
<td>This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.</td>
</tr>
<tr>
<td>TOnGetRecordCountEvent</td>
<td>This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.</td>
</tr>
<tr>
<td>TOnModifyRecordEvent</td>
<td>This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.</td>
</tr>
</tbody>
</table>

5.25.1 Classes

Classes in the VirtualDataSet unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomVirtualDataSet</td>
<td>A base class for representation of arbitrary data in tabular form.</td>
</tr>
<tr>
<td>TVirtualDataSet</td>
<td>Dataset that processes arbitrary non-tabular data.</td>
</tr>
</tbody>
</table>
5.25.1.1 TCustomVirtualDataSet Class

A base class for representation of arbitrary data in tabular form.

For a list of all members of this type, see TCustomVirtualDataSet members.

Unit

virtualDataSet

Syntax

TCustomVirtualDataSet = class(TMemDataSet);

Inheritance Hierarchy

TMemDataSet
  TCustomVirtualDataSet

© 1997-2021 Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback

5.25.1.1.1 Members

TCustomVirtualDataSet class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>(inherited from TMemDataSet) Used to enable or disable the use of cached</td>
</tr>
<tr>
<td></td>
<td>updates for a dataset.</td>
</tr>
<tr>
<td>IndexFieldNames</td>
<td>(inherited from TMemDataSet) Used to get or set the list of fields on which</td>
</tr>
<tr>
<td></td>
<td>the recordset is sorted.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>(inherited from TMemDataSet) Specifies the upper and lower boundaries for a</td>
</tr>
<tr>
<td></td>
<td>range.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>(inherited from TMemDataSet) Used to avoid setting the Required property of</td>
</tr>
<tr>
<td></td>
<td>a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>(inherited from TMemDataSet) Used to prevent implicit update of rows on</td>
</tr>
<tr>
<td></td>
<td>database.</td>
</tr>
</tbody>
</table>
**Prepared** (inherited from **TMemDataSet**)
Determines whether a query is prepared for execution or not.

**Ranged** (inherited from **TMemDataSet**)
Indicates whether a range is applied to a dataset.

**UpdateRecordTypes** (inherited from **TMemDataSet**)
Used to indicate the update status for the current record when cached updates are enabled.

**UpdatesPending** (inherited from **TMemDataSet**)
Used to check the status of the cached updates buffer.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td><strong>Prepare</strong> (inherited from TMemDataSet)</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong> (inherited from TMemDataSet)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong> (inherited from TMemDataSet)</td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>SaveToXML</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><strong>SetRange</strong> (inherited from TMemDataSet)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong> (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong> (inherited from TMemDataSet)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong> (inherited from TMemDataSet)</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong> (inherited from TMemDataSet)</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are</td>
</tr>
</tbody>
</table>
### UpdateStatus (inherited from TMemDataSet)

Indicates the current update status for the dataset when cached updates are enabled.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnUpdateError (inherited from TMemDataSet)</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord (inherited from TMemDataSet)</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

### TVirtualDataSet Class

Dataset that processes arbitrary non-tabular data.

For a list of all members of this type, see `TVirtualDataSet` members.

#### Unit

`virtualDataSet`

#### Syntax

```delphi
TVirtualDataSet = class(TCustomVirtualDataSet);
```

#### Inheritance Hierarchy

- `TMemDataSet`
  - `TCustomVirtualDataSet`
  - `TVirtualDataSet`
5.25.1.2.1 Members

**TVirtualDataSet** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdatesPending</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>) Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>CancelRange</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>GetBlob</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Locate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LocateEx</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Prepare</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>RestoreUpdates</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>RevertRecord</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SetRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>SetRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><strong>UpdateResult</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td><strong>UpdateStatus</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnUpdateError</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td><strong>OnUpdateRecord</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>
## 5.25.2 Types

Types in the `VirtualDataSet` unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TOnDeleteRecordEvent</code></td>
<td>This type is used for the <code>E:Devart.Dac.TVirtualDataSet.OnDeleteRecord</code> event.</td>
</tr>
<tr>
<td><code>TOnGetFieldValueEvent</code></td>
<td>This type is used for the <code>E:Devart.Dac.TVirtualDataSet.OnGetFieldValue</code> event.</td>
</tr>
<tr>
<td><code>TOnGetRecordCountEvent</code></td>
<td>This type is used for the <code>E:Devart.Dac.TVirtualDataSet.OnGetRecordCount</code> event.</td>
</tr>
<tr>
<td><code>TOnModifyRecordEvent</code></td>
<td>This type is used for <code>E:Devart.Dac.TVirtualDataSet.OnInsertRecord</code> and <code>E:Devart.Dac.TVirtualDataSet.OnModifyRecord</code> events.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

### 5.25.2.1 TOnDeleteRecordEvent Procedure Reference

This type is used for the `E:Devart.Dac.TVirtualDataSet.OnDeleteRecord` event.

**Unit**

`virtualDataSet`

**Syntax**

```pascal
TOnDeleteRecordEvent = procedure (Sender: TObject; RecNo: Integer) of object;
```

**Parameters**

- **Sender**
  - An object that raised the event.
- **RecNo**
  - Number of the record being deleted.
5.25.2.2 TOnGetFieldValueEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.

Unit

virtualDataSet

Syntax

TOnGetFieldValueEvent = procedure (Sender: TObject; Field: TField; RecNo: Integer; out Value: Variant) of object;

Parameters

Sender
An object that raised the event.

Field
The field, which data has to be returned.

RecNo
The number of the record, which data has to be returned.

Value
Requested field value.

5.25.2.3 TOnGetRecordCountEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.

Unit

virtualDataSet

Syntax

TOnGetRecordCountEvent = procedure (Sender: TObject; out Count: Integer) of object;

Parameters

Sender
An object that raised the event.
Count
The number of records that the virtual dataset will contain.

5.25.2.4 TOnModifyRecordEvent Procedure Reference

This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.

Unit
virtualDataSet

Syntax

TOnModifyRecordEvent = procedure (Sender: TObject; var RecNo: Integer) of object;

Parameters

Sender
An object that raised the event.

RecNo
Number of the record being inserted or modified.

5.26 VirtualTable

This unit contains implementation of the TVirtualTable component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVirtualTable</td>
<td>Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.</td>
</tr>
</tbody>
</table>

© 2017-2021 Devart. All Rights Reserved.
5.26.1 Classes

Classes in the VirtualTable unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVirtualTable</td>
<td>Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.</td>
</tr>
</tbody>
</table>

5.26.1.1 TVirtualTable Class

Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

For a list of all members of this type, see TVirtualTable members.

Unit
virtualTable

Syntax

```pascal
TVirtualTable = class(TMemDataSet);
```

Inheritance Hierarchy

TMemDataSet
TVirtualTable

Members

TVirtualTable class overview.

Properties
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates (inherited from TMemDataSet)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td>DefaultSortType</td>
<td>Used to determine the default type of local sorting for string fields.</td>
</tr>
<tr>
<td>IndexFieldNames (inherited from TMemDataSet)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td>KeyExclusive (inherited from TMemDataSet)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>LocalConstraints (inherited from TMemDataSet)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td>LocalUpdate (inherited from TMemDataSet)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>Prepared (inherited from TMemDataSet)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged (inherited from TMemDataSet)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>UpdateRecordTypes (inherited from TMemDataSet)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td>UpdatesPending (inherited from TMemDataSet)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyRange (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>Assign</td>
<td>Copies fields and data from another TDataSet component.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from TMemDataSet)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from TMemDataSet)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from TMemDataSet)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from TMemDataSet)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from TMemDataSet)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from TMemDataSet)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>LoadFromFile</strong></td>
<td>Loads data from a file into a TVirtualTable component.</td>
</tr>
<tr>
<td><strong>LoadFromStream</strong></td>
<td>Copies data from a stream into a TVirtualTable component.</td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Excludes features that don’t need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td><strong>Prepare</strong> (inherited from TMemDataSet)</td>
<td>Allocates resources and creates field components for a dataset.</td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong> (inherited from TMemDataSet)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><strong>RevertRecord</strong> (inherited from TMemDataSet)</td>
<td>Cancels changes made to the current record when</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>SaveToXML</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><code>SetRange</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><code>SetRangeEnd</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><code>SetRangeStart</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><code>UnPrepare</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><code>UpdateResult</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td><code>UpdateStatus</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OnUpdateError</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td><code>OnUpdateRecord</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>
5.26.1.2 Properties

Properties of the `TVirtualTable` class.

For a complete list of the `TVirtualTable` class members, see the [TVirtualTable Members](#) topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CachedUpdates</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><code>IndexFieldNames</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><code>KeyExclusive</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><code>LocalConstraints</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.</td>
</tr>
<tr>
<td><code>LocalUpdate</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><code>Prepared</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><code>Ranged</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td><code>UpdateRecordTypes</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to indicate the update status for the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><code>UpdatesPending</code> (inherited from <code>TMemDataSet</code>)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
</tbody>
</table>
### DefaultSortType Property

Used to determine the default type of local sorting for string fields.

**Class**

**TVirtualTable**

**Syntax**

```plaintext
property DefaultSortType: TSortType default stCaseSensitive;
```

**Remarks**

The DefaultSortType property is used when a sort type is not specified explicitly after the field name in the `TMemDataSet.IndexFieldNames` property of a dataset.

---

### Methods

Methods of the **TVirtualTable** class.

For a complete list of the **TVirtualTable** class members, see the **TVirtualTable Members** topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyRange</strong></td>
<td>Applies a range to the</td>
</tr>
<tr>
<td>(inherited from <strong>TMemDataSet</strong>)</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>Assign</strong></td>
<td>Copies fields and data from another TDataSet component.</td>
</tr>
<tr>
<td><strong>CancelRange</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td><strong>CancelUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>DeferredPost</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td><strong>EditRangeStart</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td><strong>GetBlob</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.</td>
</tr>
<tr>
<td><strong>LoadFromFile</strong></td>
<td>Loads data from a file into a TVirtualTable component.</td>
</tr>
<tr>
<td><strong>LoadFromStream</strong></td>
<td>Copies data from a stream into a TVirtualTable component.</td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td><strong>LocateEx</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td><strong>Prepare</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Allocates resources and</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>RestoreUpdates</code></td>
<td>Marks all records in the cache of updates as unapplied.</td>
</tr>
<tr>
<td><code>RevertRecord</code></td>
<td>Cancels changes made to the current record when cached updates are enabled.</td>
</tr>
<tr>
<td><code>SaveToXML</code></td>
<td>Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.</td>
</tr>
<tr>
<td><code>SetRange</code></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><code>SetRangeEnd</code></td>
<td>Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><code>SetRangeStart</code></td>
<td>Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.</td>
</tr>
<tr>
<td><code>UnPrepare</code></td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td><code>UpdateResult</code></td>
<td>Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.</td>
</tr>
<tr>
<td><code>UpdateStatus</code></td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

See Also
- TVirtualTable Class
- TVirtualTable Class Members
5.26.1.1.3.1  Assign Method

Copies fields and data from another TDataSet component.

Class

**TVirtualTable**

Syntax

```pascal
procedure Assign(Source: TPersistent); override;
```

Parameters

**Source**

Holds the TDataSet component to copy fields and data from.

Remarks

Call the Assign method to copy fields and data from another TDataSet component.

**Note:** Unsupported field types are skipped (i.e. destination dataset will contain less fields than the source one). This may happen when Source is not a TVirtualTable component but some server-oriented dataset.

Example

```pascal
Query1.SQL.Text := 'SELECT * FROM DEPT';
Query1.Active := True;
VirtualTable1.Assign(Query1);
VirtualTable1.Active := True;
```

See Also

- **TVirtualTable**

5.26.1.1.3.2  LoadFromFile Method

Loads data from a file into a TVirtualTable component.

Class
**TVirtualTable**

**Syntax**

```plaintext
procedure LoadFromFile(const FileName: string; LoadFields: boolean = True; DecodeHTMLEntities: boolean = True);
```

**Parameters**

- `FileName`<br>  Holds the name of the file to load data from.
- `LoadFields`<br>  Indicates whether to load fields from the file.
- `DecodeHTMLEntities`<br>  Indicates whether to decode HTML entities from the file.

**Remarks**

Call the `LoadFromFile` method to load data from a file into a `TVirtualTable` component. Specify the name of the file to load into the field as the value of the `FileName` parameter. This file may be an XML document in ADO-compatible format or in virtual table data format. The file format is detected automatically.

**5.26.1.3.3  LoadFromStream Method**

Copies data from a stream into a `TVirtualTable` component.

**Class**

`TVirtualTable`

**Syntax**

```plaintext
procedure LoadFromStream(Stream: TStream; LoadFields: boolean = True; DecodeHTMLEntities: boolean = True);
```

**Parameters**

- `Stream`<br>  Holds the stream from which the field's value is copied.
- `LoadFields`<br>  Indicates whether to load fields from the stream.
- `DecodeHTMLEntities`
Indicates whether to decode HTML entities from the stream.

Remarks

Call the LoadFromStream method to copy data from a stream into a TVirtualTable component. Specify the stream from which the field's value is copied as the value of the Stream parameter. Data in the stream may be in ADO-compatible format or in virtual table data format. The data format is detected automatically.