# Table of Contents

**Part I  What's New**  
1

**Part II  General Information**  
15

1 Overview .......................................................................................................................... 15
2 Features ............................................................................................................................ 19
3 Requirements .................................................................................................................... 22
4 Compatibility ..................................................................................................................... 23
5 Using Several DAC Products in One IDE ........................................................................ 28
6 Component List .................................................................................................................. 28
7 Hierarchy Chart ................................................................................................................ 30
8 Editions ............................................................................................................................... 31
9 Licensing ............................................................................................................................. 34
10 Getting Support .................................................................................................................. 37
11 Frequently Asked Questions ............................................................................................. 38

**Part III  Getting Started**  
47

1 Installation ........................................................................................................................... 47
2 Migration Wizard .................................................................................................................. 50
3 Migration from BDE ............................................................................................................. 51
4 Connecting to MySQL Server ............................................................................................. 54
5 Connecting to MySQL Embedded ....................................................................................... 58
6 Creating Database Objects ................................................................................................ 65
7 Deleting Data From Tables ................................................................................................ 69
8 Inserting Data Into Tables .................................................................................................. 73
9 Retrieving Data .................................................................................................................... 78
10 Modifying Data .................................................................................................................... 80
11 Stored Procedures .............................................................................................................. 84
12 Working With Result Sets Using Stored Procedures ....................................................... 90
13 Demo Projects ................................................................................................................... 92
14 Deployment ........................................................................................................................ 99

**Part IV  Using MyDAC**  
100

1 Updating Data with MyDAC Dataset Components ............................................................ 101
2 Master/Detail Relationships ............................................................................................... 101
3 Data Type Mapping ............................................................................................................. 103
4 Data Encryption .................................................................................................................... 110
5 Working in an Unstable Network ......................................................................................... 112
# Part V Reference

## 6 Disconnected Mode

- Source Property
- ProblemCount Property
- MovedCount Property
- Mode Property
- Mappings Property
- FieldMappingMode Property
- Destination Property
- CommitCount Property
- ChangedCount Property
- AbortOnProblem Property
- AbortOnKeyViol Property
- ChangedCount Property
- CommitCount Property
- Destination Property
- FieldMappingMode Property
- KeyViolCount Property
- Mappings Property
- Mode Property
- MovedCount Property
- ProblemCount Property
- RecordCount Property
- Source Property

## 7 Secure Connections

- Connecting via SSL
- Connecting via SSH
- Network Tunneling

## 8 Batch Operations

## 9 Increasing Performance

## 10 Macros

## 11 DataSet Manager

## 12 Embedded Server

## 13 National Characters

## 14 Connection Pooling

## 15 DBMonitor

## 16 Writing GUI Applications with MyDAC

## 17 Compatibility with Previous Versions

## 18 64-bit Development with Embarcadero RAD Studio XE2

## 19 Database Specific Aspects of 64-bit Development
3 CREncryption ................................................................. 184
   Classes ........................................................................... 184
      TCRCryptor Class ....................................................... 185
         Members ....................................................................... 185
      Properties ....................................................................... 186
         DataHeader Property................................................... 187
         EncryptionAlgorithm Property..................................... 187
         HashAlgorithm Property............................................... 187
         InvalidHashAction Property........................................... 188
         Password Property...................................................... 189
      Methods ....................................................................... 189
         SetKey Method......................................................... 190
   Enumerations ..................................................................... 190
      TCRCryptDataHeader Enumeration...................................... 191
      TCRCryptEncryptionAlgorithm Enumeration...................... 191
      TCRCryptHashAlgorithm Enumeration............................. 192
      TCRCryptInvalidHashAction Enumeration......................... 193
4 CRGrid ................................................................. 193
   Classes ........................................................................... 193
      TCRDBGrid Class .......................................................... 194
         Members ....................................................................... 194
      Properties ....................................................................... 197
         Columns Property....................................................... 198
         Filtered Property........................................................ 198
         LevelDelimiterChar Property........................................ 199
         OnMemoClick Property............................................... 199
         OptionsEx Property.................................................... 200
         OptionsMenu Property.............................................. 201
      Methods ....................................................................... 201
         ActivateFilterEdit Method............................................. 202
         ActivateSearchEdit Method.......................................... 203
         AdjustColumns Method................................................. 203
         ApplyFilter Method..................................................... 204
         CalcTitleLevel Method............................................... 204
         ClearFilters Method..................................................... 205
         ClearSorting Method..................................................... 205
         DataChanged Method.................................................... 205
         GetGridSize Method..................................................... 206
         GetTitleLevel Method................................................. 206
      Events ......................................................................... 207
         OnGetCellParams Event............................................... 207
5 CRVio ................................................................. 207
   Classes ........................................................................... 208
      IHttpOptions Class ....................................................... 208
         Members ....................................................................... 209
6 DAAlerter ............................................................................................................................................... 217

Classes ............................................................................................................................................... 218
TDAlerter Class ............................................................................................................................... 218
Members ............................................................................................................................................... 219
Properties .............................................................................................................................................. 219
Active Property .................................................................................................................................... 220
AutoRegister Property .................................................................................................................. 220
Methods ............................................................................................................................................... 221
SendEvent Method ......................................................................................................................... 221
Start Method ......................................................................................................................................... 222
Stop Method .......................................................................................................................................... 223
Events .................................................................................................................................................. 223
OnError Event ....................................................................................................................................... 224

7 DADump ............................................................................................................................................ 224

Classes .............................................................................................................................................. 225
TDADump Class .................................................................................................................................. 225
Members ............................................................................................................................................... 226
Properties .............................................................................................................................................. 227
Connection Property ......................................................................................................................... 228
Debug Property .................................................................................................................................... 229
Options Property .............................................................................................................................. 230
SQL Property ....................................................................................................................................... 230
TableName Property .......................................................................................................................... 231
Methods ............................................................................................................................................... 231
Backup Method ............................................................................................................................... 232
BackupQuery Method ..................................................................................................................... 233
BackupToFile Method ..................................................................................................................... 234
BackupToFileStream Method ......................................................................................................... 234
Restore Method ............................................................................................................................... 235
RestoreFromFile Method ............................................................................................................... 236
RestoreFromStream Method ........................................................................................................... 236
Events .................................................................................................................................................. 237
OnBackupProgress Event .............................................................................................................. 238
OnError Event ................................................................................................................................. 238
OnRestoreProgress Event ............................................................................................................. 239
TDAProcOptions Class .................................................................................................................... 240
Members ............................................................................................................................................... 240
### TDAScript Class

- Members
- Properties
  - FieldType Property
  - Name Property
- Methods
  - CreateColumns Method
  - Load Method
  - LoadFromDataSet Method
  - PutColumnData Method
  - PutColumnData Method
- Events
  - OnGetColumnData Event
  - OnProgress Event
  - OnPutData Event

### DALoader Class

- Members
- Properties
  - Columns Property
  - Connection Property
  - TableName Property
- Methods
  - CreateColumns Method
  - Load Method
  - LoadFromDataSet Method
  - PutColumnData Method
  - PutColumnData Method
- Events
  - OnGetColumnData Event
  - OnProgress Event
  - OnPutData Event

### DALoaderOptions Class

- Members
- Properties
  - UseBlankValues Property

### TDAScript Class

- Members
- Properties
  - Connection Property
  - DataSet Property
  - Debug Property
  - Delimiter Property

### DALoader Class

- Members
- Properties
  - Columns Property
  - Connection Property
  - TableName Property
- Methods
  - CreateColumns Method
  - Load Method
  - LoadFromDataSet Method
  - PutColumnData Method
  - PutColumnData Method
- Events
  - OnGetColumnData Event
  - OnProgress Event
  - OnPutData Event
10 DASQL Monitor

Classes

10.1 TCustomDASQLMonitor Class

Members

Properties

Methods

Enumerations

TErrorAction Enumeration

© 2021 Devart
11 DBAccess ........................................................................................................................................... 311

Classes ...................................................................................................................................................... 315
EDAError Class ........................................................................................................................................ 317
  Members .............................................................................................................................................. 317
  Properties .......................................................................................................................................... 318
    Component Property ..................................................................................................................... 318
    ErrorCode Property .................................................................................................................... 319
TCRDataSource Class .............................................................................................................................. 319
  Members .............................................................................................................................................. 320
TCustomConnectDialog Class .................................................................................................................. 320
  Members .............................................................................................................................................. 320
  Properties .......................................................................................................................................... 322
    CancelButton Property .................................................................................................................. 323
    Caption Property ........................................................................................................................ 323
    ConnectButton Property .............................................................................................................. 323
    DialogClass Property .................................................................................................................. 324
    LabelSet Property ........................................................................................................................ 325
    PasswordLabel Property .............................................................................................................. 325
    Retries Property .......................................................................................................................... 325
    SavePassword Property .............................................................................................................. 326
    ServerLabel Property ................................................................................................................ 326
    StoreLogInfo Property .............................................................................................................. 327
    UsernameLabel Property ............................................................................................................ 327
  Methods .............................................................................................................................................. 328
    Execute Method ........................................................................................................................ 328
    GetServerList Method ................................................................................................................ 329
TCustomDAConnection Class .................................................................................................................... 329
  Members .............................................................................................................................................. 330
  Properties .......................................................................................................................................... 332
    ConnectDialog Property ............................................................................................................. 333
    ConnectString Property ............................................................................................................ 334
    ConvertEOL Property ................................................................................................................ 336
    InTransaction Property ............................................................................................................ 336
    LoginPrompt Property .............................................................................................................. 337
    Options Property ....................................................................................................................... 337
    Password Property ..................................................................................................................... 338
### TCustomDADataSet Class

<table>
<thead>
<tr>
<th>Members</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Members

- **RowsAffected Property**
- **RefreshOptions Property**
- **Params Property**
- **ParamCheck Property**
- **Options Property**
- **Macros Property**
- **MacroCount Property**
- **KeyFields Property**
- **IsQuery Property**
- **FinalSQL Property**
- **FilterSQL Property**
- **Disconnected Property**
- **Debug Property**
- **Connection Property**
- **Conditions Property**
- **DataTypeMap Property**
- **Debug Property**
- **DetailFields Property**
- **Disconnected Property**
- **FetchRows Property**
- **FilterSQL Property**
- **FinalSQL Property**
- **IsQuery Property**
- **KeyFields Property**
- **MacroCount Property**
- **Macros Property**
- **MasterFields Property**
- **MasterSource Property**
- **Options Property**
- **ParamCheck Property**
- **ParamCount Property**
- **Params Property**
- **ReadOnly Property**
- **RefreshOptions Property**
- **RowsAffected Property**

#### Methods

- **StartTransaction Method**
- **RemoveFromPool Method**
- **Ping Method**
- **GetStoredProcNames Method**
- **GetKeyFieldNames Method**
- **ExecSQL Method**
- **ExecProcEx Method**
- **ExecProc Method**
- **Disconnect Method**
- **CreateSQL Method**
- **Connect Method**
- **ApplyUpdates Method**

#### Events

- **OnConnectionLost Event**
- **OnError Event**

#### Properties

- **Pooling Property**
- **PoolingOptions Property**
- **Server Property**
- **Username Property**

© 2021 Devart
TCustomDASQL Class

Properties

Members

Events

Methods

SQL Property .................................................................................................................. 388
SQLDelete Property ......................................................................................................... 389
SQLInsert Property ......................................................................................................... 390
SQLLock Property ........................................................................................................... 390
SQLRecCount Property ................................................................................................... 391
SQLRefresh Property ....................................................................................................... 392
SQLUpdate Property ........................................................................................................ 393
UniDirectional Property .................................................................................................. 394

AddWhere Method ......................................................................................................... 398
BreakExec Method ........................................................................................................... 399
CreateBlobStream Method ............................................................................................... 400
DeleteWhere Method ...................................................................................................... 400
Execute Method ............................................................................................................... 401
Execute Method .............................................................................................................. 401
Execute Method .............................................................................................................. 401
Executing Method ............................................................................................................ 402
Fetch Method ................................................................................................................... 403
Fetching Method ............................................................................................................... 404
FetchingAll Method ......................................................................................................... 404
FindKey Method .............................................................................................................. 405
FindMacro Method .......................................................................................................... 405
FindNearest Method ........................................................................................................ 406
FindParam Method .......................................................................................................... 407
GetDataType Method ....................................................................................................... 408
GetFieldObject Method ................................................................................................... 408
GetFieldPrecision Method ............................................................................................... 409
GetFieldScale Method .................................................................................................... 410
GetKeyFieldNames Method .............................................................................................. 410
GetOrderBy Method ........................................................................................................ 411
GotoCurrent Method ....................................................................................................... 412
Lock Method .................................................................................................................... 412
MacroByName Method .................................................................................................... 413
ParamByName Method ..................................................................................................... 414
Prepare Method ............................................................................................................... 415
RefreshRecord Method .................................................................................................... 415
RestoreSQL Method ........................................................................................................ 416
SaveSQL Method ............................................................................................................. 417
SetOrderBy Method ......................................................................................................... 417
SQLSaved Method .......................................................................................................... 418
Unlock Method ................................................................................................................. 418

Events

AfterExecute Event ......................................................................................................... 420
AfterFetch Event .............................................................................................................. 420
AfterUpdateExecute Event ............................................................................................... 421
BeforeFetch Event ........................................................................................................... 421
BeforeUpdateExecute Event ............................................................................................. 422

TCustomDASQL Class

Members ............................................................................................................................ 422

Properties

ChangeCursor Property .................................................................................................... 425
Connection Property ....................................................................................................... 426
Debug Property ............................................................................................................... 427
FinalSQL Property .......................................................................................................... 428

© 2021 Devart
TDAConditions Class

Members

Properties

Methods

Enable Method
Value Property
Name Property
Enabled Property
ExecSQL Method
Apply Method

SQL Property

TCustomDAUpdateSQL Class

Members

Properties

Methods

Execute Method
Execute Method
Execute Method
Executing Method
FindMacro Method
FindParam Method
MacroByName Method
ParamByName Method
Prepare Method
UnPrepare Method
WaitExecuting Method

AfterExecute Event

Events

Params Property
ParamCount Property
Macros Property
MacroCount Property

TDACondition Class

Members

Properties

Enabled Property
Name Property
Value Property

Methods

Disable Method
Enable Method

TDAConditions Class

Members

Properties
Condition Property(Indexer)........................................................................................................... 461
Enabled Property.......................................................................................................................... 462
Item Property(Indexer).................................................................................................................. 462
Text Property................................................................................................................................ 463
WhereSQL Property....................................................................................................................... 463
Methods ........................................................................................................................................ 463
Add Method .................................................................................................................................... 464
Add Method .................................................................................................................................... 465
Add Method .................................................................................................................................... 465
Delete Method .................................................................................................................................. 466
Disable Method ............................................................................................................................... 466
Enable Method .................................................................................................................................. 467
Find Method .................................................................................................................................... 467
Get Method ..................................................................................................................................... 467
IndexOf Method .............................................................................................................................. 468
Remove Method .............................................................................................................................. 468
TDAConnectionOptions Class ........................................................................................................ 469
Members ...................................................................................................................................... 469
Properties ..................................................................................................................................... 470
Allow ImplicitConnect Property ...................................................................................................... 471
DefaultSortType Property ............................................................................................................. 472
DisconnectedMode Property .......................................................................................................... 472
KeepDesignConnected Property .................................................................................................... 473
LocalFailover Property .................................................................................................................. 473
TDAConnectionSSLOptions Class ..................................................................................................... 474
Members ...................................................................................................................................... 474
Properties ..................................................................................................................................... 474
CA_CERT Property .......................................................................................................................... 475
cert Property .................................................................................................................................. 475
CipherList Property .......................................................................................................................... 476
Key Property ................................................................................................................................... 476
TDADataSetOptions Class ................................................................................................................ 477
Members ...................................................................................................................................... 477
Properties ..................................................................................................................................... 480
AutoPrepare Property ...................................................................................................................... 482
CacheCalcFields Property ............................................................................................................. 483
CompressBlobMode Property .......................................................................................................... 483
DefaultValues Property .................................................................................................................. 484
DetailDelay Property ...................................................................................................................... 484
FieldsOrigin Property ..................................................................................................................... 485
FlatBuffers Property ....................................................................................................................... 485
InsertAllSetFields Property ........................................................................................................... 486
LocalMasterDetail Property ........................................................................................................... 486
LongStrings Property ..................................................................................................................... 487
MasterFieldsNonNull Property ........................................................................................................ 487
NumberRange Property ................................................................................................................. 487
QueryRecCount Property ............................................................................................................... 488
QuoteNames Property .................................................................................................................... 488
RemoveOnRefresh Property............................................................................................................ 489
RequiredFields Property ................................................................................................................ 489
ReturnParams Property .................................................................................................................. 490
SetFieldsReadOnly Property ......................................................................................................... 490
StrictUpdate Property .................................................................................................................... 491
UpdateAllFields Property .............................................................................................................. 491
### TDAEncryption Class

**Members**

- UpdateBatchSize Property

**Properties**

- Encryptor Property
- Connection Property
- IgnoreErrors Property
- FieldType Property
- FieldLength Property
- FieldScale Property
- FieldName Property
- DBScaleMax Property
- DBScaleMin Property
- DbType Property
- Fields Property
- Restrictions Property
- metaDataKind Property
- DBLengthMax Property
- DBLengthMin Property
- DBScaleMax Property
- DBScaleMin Property
- DBType Property
- DataType Property
- AsLargeInt Property
- AsFloat Property
- AsBlobRef Property
- AsMemo Property
- AsMemoRef Property
- AsSQLTimeStamp Property
- AsInt Property
- AsInteger Property
- AsString Property
- As WideString Property
- AsFloat Property
- AsBlob Property
- AsLargeInt Property
- AsMemo Property
- AsMemoRef Property
- AsSQLTimeStamp Property
- IsNull Property
- ParamType Property
- Size Property
- Value Property

### TDAMapRule Class

**Members**

- LoadFromFile Method
- AssignField Method
- Value Property

**Properties**

- DbType Property
- DBScaleMax Property
- DBScaleMin Property
- DBType Property
- DBLengthMax Property
- DBLengthMin Property
- metaDataKind Property
- Connection Property
- Restrictions Property
- IgnoreErrors Property
- IgnoreInvalidRules Property
- Encryptor Property

### TDAMapRules Class

**Members**

- UpdateBatchSize Property

**Properties**

- metaDataKind Property
- Connection Property
- Restrictions Property
- IgnoreErrors Property
- IgnoreInvalidRules Property
- Encryptor Property

### TDAMetaData Class

**Members**

- LoadFromFile Method
- AssignField Method
- Value Property

**Properties**

- DbType Property
- DBScaleMax Property
- DBScaleMin Property
- DBType Property
- DBLengthMax Property
- DBLengthMin Property
- metaDataKind Property
- Connection Property
- Restrictions Property
- IgnoreErrors Property
- Encryptor Property

---

© 2021 Devart
<table>
<thead>
<tr>
<th>Class</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAParams Class</td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td>Items Property (Indexer)</td>
</tr>
<tr>
<td></td>
<td>Methods</td>
</tr>
<tr>
<td></td>
<td>FindParam Method</td>
</tr>
<tr>
<td></td>
<td>ParamByName Method</td>
</tr>
<tr>
<td>TDA Transaction Class</td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td>Active Property</td>
</tr>
<tr>
<td></td>
<td>DefaultCloseAction Property</td>
</tr>
<tr>
<td></td>
<td>Methods</td>
</tr>
<tr>
<td></td>
<td>Commit Method</td>
</tr>
<tr>
<td></td>
<td>Rollback Method</td>
</tr>
<tr>
<td></td>
<td>StartTransaction Method</td>
</tr>
<tr>
<td></td>
<td>Events</td>
</tr>
<tr>
<td></td>
<td>OnCommit Event</td>
</tr>
<tr>
<td></td>
<td>OnCommitRetaining Event</td>
</tr>
<tr>
<td></td>
<td>OnAbort Event</td>
</tr>
<tr>
<td></td>
<td>OnRollback Retaining Event</td>
</tr>
<tr>
<td>TMacro Class</td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td>Active Property</td>
</tr>
<tr>
<td></td>
<td>AsDateTime Property</td>
</tr>
<tr>
<td></td>
<td>AsFloat Property</td>
</tr>
<tr>
<td></td>
<td>AsInteger Property</td>
</tr>
<tr>
<td></td>
<td>AsString Property</td>
</tr>
<tr>
<td></td>
<td>Name Property</td>
</tr>
<tr>
<td></td>
<td>Value Property</td>
</tr>
<tr>
<td>TMacros Class</td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td>Items Property (Indexer)</td>
</tr>
<tr>
<td></td>
<td>Methods</td>
</tr>
<tr>
<td></td>
<td>AssignValues Method</td>
</tr>
<tr>
<td></td>
<td>Expand Method</td>
</tr>
<tr>
<td></td>
<td>FindMacro Method</td>
</tr>
<tr>
<td></td>
<td>IsEqual Method</td>
</tr>
<tr>
<td></td>
<td>MacroByName Method</td>
</tr>
<tr>
<td></td>
<td>Scan Method</td>
</tr>
<tr>
<td>TPoolingOptions Class</td>
<td>Members</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td>ConnectionLifetime Property</td>
</tr>
<tr>
<td></td>
<td>MaxPoolSize Property</td>
</tr>
<tr>
<td></td>
<td>MinPoolSize Property</td>
</tr>
<tr>
<td></td>
<td>Validate Property</td>
</tr>
</tbody>
</table>

© 2021 Devart
12 MemData

Classes

TAttribute Class

Members

Properties

Attributes Property

DataType Property

Length Property

ObjectType Property

Owner Property

Scale Property

Size Property

Methods

AsString Method

AsWideString Method

IsUnicode Method

SaveToFile Method

SaveToStream Method

TBlob Class

Members

Properties

AsString Property

AsWideString Property

IsUnicode Property

Size Property

Methods

Assign Method

Clear Method

LoadFromFile Method

LoadFromStream Method

Read Method

SaveToFile Method

SaveToStream Method
XVI MySQL Data Access Components

ApplyUpdates Method .............................................................. 629
ApplyUpdates Method .............................................................. 630
CancelRange Method .............................................................. 631
CancelUpdates Method ............................................................ 632
CommitUpdates Method ........................................................... 632
DeferredPost Method ............................................................. 633
EditRangeEndMethod .............................................................. 634
EditRangeStartMethod ............................................................ 634
GetBlob Method ................................................................. 635
GetBlob Method ................................................................. 636
GetBlob Method ................................................................. 636
Locate Method ................................................................ 637
Locate Method ................................................................ 637
Locate Method ................................................................ 638
LocateEx Method ................................................................ 639
LocateEx Method ................................................................ 640
LocateEx Method ................................................................ 640
Prepare Method ................................................................. 641
RestoreUpdates Method ......................................................... 642
RevertRecord Method ............................................................ 643
SaveToXML Method .............................................................. 643
SaveToXML Method .............................................................. 644
SaveToXML Method .............................................................. 644
SetRange Method ................................................................ 645
SetRangeEndMethod .............................................................. 646
SetRangeStartMethod ............................................................. 647
UnPrepare Method ................................................................. 648
UpdateResult Method ............................................................. 648
UpdateStatus Method ............................................................ 649
Events ............................................................................. 650
OnUpdateError Event ............................................................. 650
OnUpdateRecord Event .......................................................... 651
Variables ........................................................................... 652
DoNotRaiseExceptionOnUaFail Variable ................................. 653
SendDataSetChangeEventAfterOpen Variable ............................ 653

14 MyAccess ........................................................................ 654

Classes .............................................................................. 656
TCustomMyConnection Class ................................................ 658
Members ............................................................................ 659
Properties ........................................................................... 662
ClientVersion Property ......................................................... 664
ConnectionTimeout Property ................................................ 664
Database Property ................................................................ 665
IsolationLevel Property ......................................................... 666
Options Property ................................................................ 666
ServerVersion Property ........................................................ 667
Threading Property .............................................................. 668
Methods ............................................................................ 668
AssignConnect Method ......................................................... 670
CreateDataSet Method ........................................................ 671
ExecSQL Method ................................................................. 672
GetCharsetName Method ....................................................... 672
GetExecuteInfo Method ........................................................ 673
GetTriggerNames Method ..................................................... 674
ReleaseSavepoint Method ................................................................. 675
RollbackToSavepoint Method .......................................................... 676
Savepoint Method ......................................................................... 676
TCustomMyConnectionOptions Class .................................................. 677
Members............................................................................................... 677
Properties............................................................................................. 679
Charset Property................................................................................. 680
NullForZeroDelphiDate Property....................................................... 681
NumericType Property....................................................................... 681
OptimizedBigInt Property................................................................. 682
UseUnicode Property......................................................................... 682
TCustomMyDataSet Class .................................................................. 683
Members............................................................................................... 684
Properties............................................................................................. 692
CommandTimeout Property............................................................... 697
Connection Property......................................................................... 697
Encryption Property.......................................................................... 698
FetchAll Property.............................................................................. 698
InsertId Property.............................................................................. 699
LockMode Property............................................................................ 700
Options Property.............................................................................. 700
SmartFetch Property......................................................................... 702
UpdateObject Property...................................................................... 703
Methods............................................................................................... 703
GetFieldEnum Method........................................................................ 708
Lock Method....................................................................................... 709
Lock Method....................................................................................... 709
Lock Method....................................................................................... 710
LockTable Method............................................................................ 710
OpenNext Method............................................................................. 711
RefreshQuick Method....................................................................... 712
UnLockTable Method......................................................................... 713
TCustomMyStoredProc Class ............................................................... 714
Members............................................................................................... 714
Properties............................................................................................. 724
StoredProcName Property................................................................. 728
Methods............................................................................................... 728
ExecProc Method.............................................................................. 733
PrepareSQL Method.......................................................................... 734
TCustomMyTable Class ...................................................................... 734
Members............................................................................................... 735
Properties............................................................................................. 744
IndexDefs Property........................................................................... 749
Limit Property................................................................................... 749
Offset Property............................................................................... 750
Options Property.............................................................................. 750
TableName Property.......................................................................... 751
Methods............................................................................................... 751
EmptyTable Method.......................................................................... 756
TMyCommand Class ........................................................................ 756
Members............................................................................................... 756
Properties............................................................................................. 757
CommandTimeout Property............................................................... 761
Connection Property......................................................................... 762
### TMyConnection Class

<table>
<thead>
<tr>
<th>Properties</th>
<th>Members</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected Property</td>
<td>Database Property</td>
<td>763</td>
</tr>
<tr>
<td>HTTP Options Property</td>
<td>Options Property</td>
<td>770</td>
</tr>
<tr>
<td>IOHandler Property</td>
<td>Options Property</td>
<td>771</td>
</tr>
<tr>
<td>Options Property</td>
<td>Password Property</td>
<td>772</td>
</tr>
<tr>
<td>Server Property</td>
<td>Port Property</td>
<td>774</td>
</tr>
<tr>
<td>SSL Options Property</td>
<td>Username Property</td>
<td>775</td>
</tr>
</tbody>
</table>

### TMyConnectionOptions Class

<table>
<thead>
<tr>
<th>Properties</th>
<th>Members</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Charset Property</td>
<td>Compress Property</td>
<td>780</td>
</tr>
<tr>
<td>CheckBackslashes Property</td>
<td>Compress Property</td>
<td>781</td>
</tr>
<tr>
<td>Compress Property</td>
<td>DefaultSortType Property</td>
<td>782</td>
</tr>
<tr>
<td>Direct Property</td>
<td>DisconnectMode Property</td>
<td>783</td>
</tr>
<tr>
<td>DisconnectMode Property</td>
<td>Embedded Property</td>
<td>783</td>
</tr>
<tr>
<td>Embedded Property</td>
<td>Interactive Property</td>
<td>784</td>
</tr>
<tr>
<td>Interactive Property</td>
<td>IPVersion Property</td>
<td>784</td>
</tr>
<tr>
<td>IPVersion Property</td>
<td>KeepDesignConnected Property</td>
<td>785</td>
</tr>
<tr>
<td>KeepDesignConnected Property</td>
<td>LocalFailover Property</td>
<td>785</td>
</tr>
<tr>
<td>LocalFailover Property</td>
<td>NullForZeroDelphiDate Property</td>
<td>786</td>
</tr>
<tr>
<td>NullForZeroDelphiDate Property</td>
<td>NumericType Property</td>
<td>786</td>
</tr>
<tr>
<td>NumericType Property</td>
<td>OptimizedBigInt Property</td>
<td>787</td>
</tr>
<tr>
<td>OptimizedBigInt Property</td>
<td>Protocol Property</td>
<td>787</td>
</tr>
<tr>
<td>Protocol Property</td>
<td>UseUnicode Property</td>
<td>788</td>
</tr>
<tr>
<td>UseUnicode Property</td>
<td>TMyConnectionSSLOptions Class</td>
<td>788</td>
</tr>
</tbody>
</table>

### TMyDataSetOptions Class

<table>
<thead>
<tr>
<th>Properties</th>
<th>Members</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPrepare Property</td>
<td>AutoRefresh Property</td>
<td>789</td>
</tr>
<tr>
<td>AutoRefresh Property</td>
<td>AutoRefreshInterval Property</td>
<td>798</td>
</tr>
<tr>
<td>AutoRefreshInterval Property</td>
<td>BinaryAsString Property</td>
<td>799</td>
</tr>
<tr>
<td>BinaryAsString Property</td>
<td>CheckRow Version Property</td>
<td>799</td>
</tr>
<tr>
<td>CheckRow Version Property</td>
<td>CreateConnection Property</td>
<td>800</td>
</tr>
<tr>
<td>CreateConnection Property</td>
<td>DefaultValues Property</td>
<td>800</td>
</tr>
<tr>
<td>DefaultValues Property</td>
<td>EnableBoolean Property</td>
<td>801</td>
</tr>
<tr>
<td>EnableBoolean Property</td>
<td>FieldsAsString Property</td>
<td>801</td>
</tr>
<tr>
<td>FieldsAsString Property</td>
<td>FullRefresh Property</td>
<td>802</td>
</tr>
<tr>
<td>FullRefresh Property</td>
<td>NullForZeroDate Property</td>
<td>802</td>
</tr>
<tr>
<td>NullForZeroDate Property</td>
<td>NumberRange Property</td>
<td>803</td>
</tr>
<tr>
<td>NumberRange Property</td>
<td>QueryRecCount Property</td>
<td>803</td>
</tr>
<tr>
<td>QueryRecCount Property</td>
<td>QuoteNames Property</td>
<td>804</td>
</tr>
<tr>
<td>QuoteNames Property</td>
<td>RemoveOnRefresh Property</td>
<td>805</td>
</tr>
<tr>
<td>RemoveOnRefresh Property</td>
<td>RequiredFields Property</td>
<td>805</td>
</tr>
</tbody>
</table>
ReturnParams Property ................................................................. 806
SaveBlobsAsHex Property ............................................................ 806
SetEmptyStrToNull Property ........................................................... 807
SetFieldsReadOnly Property ........................................................... 807
StrictUpdate Property ................................................................. 807
TrimFixedChar Property ............................................................... 808
TrimVarChar Property ................................................................. 808
TMyBackup Class ........................................................................ 809
Members ................................................................................. 810
TMyDACVersion Constant ............................................................ 810
TMyEncryptor Class ..................................................................... 811
Members ................................................................................. 811
TMyEncryptor Class ..................................................................... 812
Members ................................................................................. 812
TMyMetaData Class ....................................................................... 815
Members ................................................................................. 816
Properties ................................................................................. 825
FetchAll Property ....................................................................... 830
LockMode Property ...................................................................... 831
UpdatingTable Property ............................................................... 831
TMyMetaData Class ....................................................................... 832
Members ................................................................................. 833
Properties ................................................................................. 842
LockMode Property ...................................................................... 847
UpdatingTable Property ............................................................... 848
TMyTable Class ........................................................................... 848
Members ................................................................................. 849
Properties ................................................................................. 858
FetchAll Property ....................................................................... 863
LockMode Property ...................................................................... 864
OrderFields Property ................................................................... 864
TableName Property ..................................................................... 865
TMyTableOptions Class ............................................................... 865
Members ................................................................................. 866
Properties ................................................................................. 870
HandlerIndex Property ................................................................. 874
UseHandler Property ................................................................. 874
TMyTransaction Class ............................................................... 875
Members ................................................................................. 876
TMyUpdateSQL Class ................................................................... 877
Members ................................................................................. 878
Types .......................................................................................... 879
TMyUpdateExecuteEvent Procedure Reference ......................... 880
Enumerations ................................................................................ 880
TLockRecordType Enumeration .................................................... 881
TLockType Enumeration ............................................................. 881
TMyIsolationLevel Enumeration ................................................ 882
Routines .................................................................................... 883
GetServerList Procedure ............................................................. 883
Constants ................................................................................... 884
MyDACVersion Constant ............................................................ 884
15 MyBackup ............................................................................... 884
Classes ...................................................................................... 885
TMyBackup Class ......................................................................... 885
Members .................................................................................... 886
MySQL Data Access Components

Properties ........................................................................................................................................... 888
   BackupPriority Property ........................................................................................................... 889
   Connection Property .................................................................................................................. 890
   Debug Property ........................................................................................................................ 890
   Duplicates Property ..................................................................................................................... 891
   EnclosedBy Property .................................................................................................................. 892
   EscapedBy Property .................................................................................................................... 892
   Fields Property ........................................................................................................................... 893
   FieldsTerminatedBy Property .................................................................................................... 894
   IgnoreLines Property .................................................................................................................. 894
   LinesTerminatedBy Property ....................................................................................................... 895
   Local Property ............................................................................................................................ 896
   Mode Property ............................................................................................................................. 896
   Path Property ............................................................................................................................... 897
   TableNames Property .................................................................................................................. 897

Methods ........................................................................................................................................... 898
   Backup Method ............................................................................................................................ 899
   Restore Method ........................................................................................................................... 899

Events ............................................................................................................................................... 900
   OnTableMsg Event ....................................................................................................................... 900

Types ............................................................................................................................................... 901
   TMyDBTableMsgEvent Procedure Reference ............................................................................. 901

Enumerations .................................................................................................................................... 902
   TMyBackupMode Enumeration .................................................................................................... 902
   TMyBackupPriority Enumeration ............................................................................................... 903
   TMyRestoreDuplicates Enumeration ......................................................................................... 903

16 MyBuilderClient ......................................................................................................................... 904

Classes ............................................................................................................................................. 904
   TMyBuilder Class ....................................................................................................................... 904
   Members .......................................................................................................................................... 905

17 MyClasses .................................................................................................................................... 905

Classes ............................................................................................................................................. 906
   BMyError Class ............................................................................................................................ 906
   Members .......................................................................................................................................... 907
   Properties ...................................................................................................................................... 907
   LineNumber Property .................................................................................................................. 908

Enumerations .................................................................................................................................... 908
   TMyProtocol Enumeration .......................................................................................................... 908

Variables ......................................................................................................................................... 909
   __Strings65535ToMemo Variable ............................................................................................... 909

18 MyConnectionPool ...................................................................................................................... 910

Classes ............................................................................................................................................. 910
   TMyConnectionPoolManager Class ............................................................................................ 911
   Members .......................................................................................................................................... 911

19 MyDacVcl ..................................................................................................................................... 911

Classes ............................................................................................................................................. 912
   TMyConnectDialog Class ............................................................................................................. 912
   Members .......................................................................................................................................... 913
   Properties ...................................................................................................................................... 914
   Connection Property .................................................................................................................... 914
   DatabaseLabel Property .............................................................................................................. 916
   PortLabel Property ....................................................................................................................... 917
   Show Database Property .............................................................................................................. 917

© 2021 Devart
20 MyDump ............................................................................................................... 918

**Classes** ............................................................................................................. 919
- TMyDump Class .................................................................................................. 919
  - Members ........................................................................................................... 920
  - Properties ........................................................................................................ 922
    - Connection Property ...................................................................................... 923
    - Objects Property ........................................................................................... 924
    - Options Property ........................................................................................... 925
    - StoredProcNames Property ......................................................................... 926
    - TriggerNames Property ............................................................................... 926
- TMyDumpOptions Class ......................................................................................... 927
  - Members ........................................................................................................... 927
  - Properties ........................................................................................................ 928
    - AddLock Property ........................................................................................ 930
    - CommitBatchSize Property ........................................................................... 930
    - DisableKeys Property .................................................................................. 931
    - HexBlob Property ......................................................................................... 931
    - InsertType Property ...................................................................................... 931
    - UseDelayedIns Property ............................................................................... 932
    - UseExtSyntax Property ................................................................................ 932

**Types** .................................................................................................................. 933
- TMyDumpObjects Set ................................................................................................ 933

**Enumerations** ...................................................................................................... 934
- TMyDumpObject Enumeration ............................................................................... 934

21 MyEmbConnection ................................................................................................. 935

**Classes** ................................................................................................................. 935
- TMyEmbConnection Class .................................................................................... 935
  - Members ........................................................................................................... 936
  - Properties ........................................................................................................ 940
    - BaseDir Property ........................................................................................ 942
    - DataDir Property ......................................................................................... 942
    - Params Property ........................................................................................... 943
  - Events .............................................................................................................. 944
    - OnLog Event ................................................................................................. 945
    - OnLogError Event ......................................................................................... 945

22 MyLoader ................................................................................................................. 946

**Classes** ................................................................................................................. 947
- TMyColumn Class .................................................................................................. 947
  - Members ........................................................................................................... 948
- TMyLoader Class .................................................................................................... 948
  - Members ........................................................................................................... 949
  - Properties ........................................................................................................ 950
    - Connection Property ...................................................................................... 951
    - DuplicateKeys Property ............................................................................... 952
    - Options Property ........................................................................................... 952
    - RowsPerPageQuery Property ....................................................................... 953

**Types** .................................................................................................................. 953
- TMyLoaderOptions Set .......................................................................................... 954

**Enumerations** ...................................................................................................... 954
- TMyDuplicateKeys Enumeration ......................................................................... 954
- TMyLoaderOption Enumeration ......................................................................... 955

23 MyScript .................................................................................................................. 955
## MySQL Data Access Components

### 24 MyServerControl

**Classes**
- TMyServerControl Class
- TMyScript Class

**Properties**
- Connection Property
- Debug Property
- TableNames Property
- Variables Property

**Methods**
- AnalyzeTable Method
- CheckTable Method
- CreateDatabase Method
- DropDatabase Method
- Flush Method
- GetServiceNames Method
- KillProcess Method
- OptimizeTable Method
- RepairTable Method
- ServiceStart Method
- ServiceStatus Method
- ServiceStop Method
- ShowProcessList Method
- Show Status Method
- Show Variables Method

### 25 MySqlApi

**Types**
- TMyLogEvent Procedure Reference

**Variables**
- MySQLClientLibrary Variable

### 26 MySQLMonitor

**Classes**
- TMySQLMonitor Class

**Properties**
- Connection Property
- UseOptimization Property

### 27 VirtualDataSet

**Classes**
- TCustumVirtualDataSet Class

**Types**
- TOnDeleteRecordEvent Procedure Reference
- TOnGetFieldValueEvent Procedure Reference
- TOnGetRecordCountEvent Procedure Reference
- TOnModifyRecordEvent Procedure Reference

---

© 2021 Devart
1 What's New

02-Mar-21 New Features in MyDAC 10.4:
- RAD Studio 10.4.2 Sydney is supported
- macOS 11 Big Sur is supported
- iOS 14 is supported
- Android 11 is supported
- The LastWarningCount property in TMyConnection is added
- The OnWarning event in TMyConnection 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 MyDAC 10.3:
- 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 MyDAC 10.2:
- RAD Studio 10.4 Sydney is supported
- Lazarus 2.0.8 is supported
- macOS 64-bit in Lazarus is supported

26-Nov-2019 New Features in MyDAC 10.1:
- Android 64-bit is supported
- Lazarus 2.0.6 is supported
- OpenSSL 1.1 library is supported
- Now Trial edition for macOS and Linux is fully functional
- A package containing the TMySSHIOHandler and TMySSLIOHandler components for Lazarus is added
- The TMyConnection.GetStoredProcNames and TMyConnection.GetTriggerNames methods are added

22-Jul-2019 New Features in MyDAC 10.0:
- 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
24-Jun-2019 New Features in MyDAC 9.4:
- Lazarus 2.0.2 is supported
- Performance of TMyDump for data dumping from a query is improved
- Performance of describing stored procedures is improved
- Multi-threading support is improved
- The DefaultSortType property for TVirtualTable is added
- Performance of the SaveToFile/LoadFromFile methods of TVirtualTable is significantly increased

26-Nov-2018 New Features in MyDAC 9.3:
- RAD Studio 10.3 Rio is supported
- Support for PAM and Windows authentications is added
- Support of UPPER and LOWER functions for Unified SQL is added
- TMyConnection.OnDialogAuthPlugin event is added

09-Jul-2018 New Features in MyDAC 9.2:
- Lazarus 1.8.4 is supported
- MySQL 8 is supported
- Support for sha2_password, caching_sha2_password authentications is added
- Performance of batch operations is improved
- Demo projects for IntraWeb 14 are added

19-Sep-2017 New Features in MyDAC 9.1:
- Azure Database for MySQL is supported
- JSON data type is supported

05-Apr-2017 New Features in MyDAC 9.0:
- 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
- Support for HTTPS protocol is added

25-Apr-16 New Features in MyDAC 8.7:
- RAD Studio 10.1 Berlin is supported
- Lazarus 1.6 and FPC 3.0.0 is supported
• Support for the BETWEEN statement in TDADataset.Filter is added
• Support for utf8mb4 charset is added
• SmartFetch mode in Disconnected mode is supported
• Data Type Mapping performance is improved
• Performance of TDALoader on loading data from TDataSet is improved

09-Sep-15 New Features in MyDAC 8.6:
• RAD Studio 10 Seattle is supported
• Now Trial for Win64 is a fully functional Professional Edition
• INSERT, UPDATE and DELETE batch operations are supported
• MariaDB Embedded is supported

14-Apr-15 New Features in MyDAC 8.5:
• RAD Studio XE8 is supported
• AppMethod is supported

15-Sep-14 New Features in MyDAC 8.4:
• RAD Studio XE7 is supported
• Lazarus 1.2.4 is supported
• The TCustomDADataset.GetKeyFieldNames method is added
• The ConstraintColumns metadata kind for the TMyMetaData component is added
• Demo projects for FastReport 5 are added

29-Apr-14 New Features in MyDAC 8.3:
• RAD Studio 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
• The TMyDataSetOptions.MasterFieldsNullable property is added
• Now update queries inside TDataSet descendants have correct owner

25-Dec-13 New Features in MyDAC 8.2:
• 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
• Compatibility of migrating floating-point fields from other components is improved
18-Sep-13 New Features in MyDAC 8.1:
- RAD Studio XE5 is supported
- Application development for Android is supported
- Lazarus 1.0.12 is supported
- IPv6 protocol support is added
- MariaDB 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
- 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 AllowImplicitConnect option for the TMyConnection component is added
- The SQLRecCount property for the TMyQuery and TMyStoredProc components is added
- The ScanParams property for the TMyScript component is added
- The RowsAffected property for the TMyScript component is added
- ConnectionTimeout is now used when disconnecting after connection loss

25-Apr-13 New Features in MyDAC 8.0:
- 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 for TimeStamp, Single and Extended fields in VirtualTable is added

12-Dec-12 New Features in MyDAC 7.6:
- Rad Studio XE3 Update 1 is now required
- C++Builder 64-bit for Windows is supported
- TMyDumpOptions.FieldsAsString property for dumping all data types as string is added

05-Sep-12 New Features in MyDAC 7.5:
• Rad Studio XE3 is supported
• Windows 8 is supported

21-Jun-12 New Features in MyDAC 7.2:
• Update 4 Hotfix 1 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
• Data Type Mapping support is added
• Data encryption in a client application is added
• The TMyEncryptor component for data encryption is added
• Calling of the TCustomDASQL.BeforeExecute event is added

23-Nov-11 New Features in MyDAC 7.1:
• Update 4 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 0.9.30.4 and FPC 2.6.0 are supported
• Mac OS X in Lazarus is supported
• Linux x64 in Lazarus is supported
• FreeBSD in Lazarus is supported
• Performance of SQL query generation for stored procedure execution is improved

15-Sep-11 New Features in MySQL Data Access Components 7.00:
• 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 MySQL Data Access Components 6.10:
• Lazarus 0.9.30 and FPC 2.4.2 is supported
• Now the BreakExec method also stops working when getting record count if QueryRecCount=True
13-Sep-10 New Features in MySQL Data Access Components 6.00:
- Embarcadero RAD Studio XE supported

10-Sep-09 New Features in MySQL Data Access Components 5.90:
- Embarcadero RAD Studio 2010 supported

02-Apr-09 New Features in MySQL Data Access Components 5.80:
- Free Pascal under Linux supported
- Added NoPreconnect property to TMyScript for executing CONNECT and CREATE DATABASE commands

23-Oct-08 New Features in MySQL Data Access Components 5.70:
- Delphi 2009 and C++Builder 2009 supported
- Extended Unicode support for Delphi 2007 added (special Unicode build)
- Free Pascal 2.2 supported
- Powerful design-time editors implemented in Lazarus
- Completed with more comprehensive structured Help

21-Aug-08 New Features in MySQL Data Access Components 5.55:
- dbForge Fusion for MySQL support added

23-May-08 New Features in MySQL Data Access Components 5.50:
- Added compatibility with UniDAC
- Improved support of default field values
- The new component for metadata receiving added
- Added ability to specify key fields for a dataset
- Added support of automatic records locking
27-Sep-07 New Features in MySQL Data Access Components 5.20:
- CodeGear RAD Studio 2007 supported
- Added the OnProgress event in TMyLoader

12-Jun-07 New Features in MySQL Data Access Components 5.10:
- C++Builder 2007 supported

22-Mar-07 New Features in MySQL Data Access Components 5.00:

New functionality:
- Delphi 2007 for Win32 supported
- Implemented Disconnected Model for working offline and automatically connecting and disconnecting
- Implemented Local Failover for detecting connection loss and implicitly re-executing some operations
- Support for SSH protocol via SecureBridge component set added
- Added DataSet Manager to control project datasets
- Integration with MyDeveloper Tools 2.00 added
- New TCRBatchMove component for transferring data between all types of TDataSet descendants added
- Output parameters from stored procedures and functions retrieval supported
- Data export and import to/from XML supported
- WideMemo field type in Delphi 2006 supported
- AutoRefresh mode support added
- Option to break long-duration query execution added
- Ability to search for installed MySQL servers on the network added
- Support for sending messages to DBMonitor from any point in your program added

Support for more MySQL server functionality:
- HANDLER syntax support in TMyTable added
- Enumeration value retrieval for ENUM and SET fields added
Extensions and improvements to existing functionality:
- General performance improved
- **Master/detail** functionality extensions:
  - **Local master/detail** relationship support added
  - Support for master/detail relationships in **CachedUpdates** mode added
- **TMyScript** component improvements:
  - Support for executing **individual statements** in scripts added
  - Support for **executing huge scripts stored in files** with dynamic loading added
  - Ability to use standard MySQL client tool syntax added
  - Working with **calculated and lookup fields** improvements:
    - Local **sorting** and filtering added
    - Record location speed increased
    - Improved working with lookup fields
    - Greatly increased **performance of applying updates** in **CachedUpdates** mode
- **Connection pool** functionality improvements:
  - Efficiency significantly improved
  - **API for draining the connection pool** added
  - Option to **ignore or replace records** with duplicated key values in **TMyLoader** added
  - Enhanced **TMyServerControl** functionality for working with **server values**
  - Ability to customize update commands by attaching external components to **TMyUpdateSQL** objects added
  - Ability to **include all fields** in automatically generated update SQLs added

Usability improvements:
- **Syntax highlighting** in design-time editors added
- Completely restructured and clearer **demo projects**

28-Aug-06 New Features in MySQL Data Access Components 4.40:
- Optimized **TDALoader.CreateColumns** method
- Support for Professional editions of Turbo Delphi, Turbo Delphi for .NET, Turbo C++ added
- Added support for macros names in which first symbol is digit
- Added capability to use quoted field names in **IndexFieldNames** property

18-May-06 New Features in MySQL Data Access
Components 4.30.1:
- MyDAC is now compatible with InterBase Data Access Components
- Modifying FieldDefs in TVirtualTable component accelerated
- Performance of SaveToFile and LoadFromFile functions in TVirtualTable improved

26-Jan-06 New Features in MySQL Data Access Components 4.30:
- Support for Delphi 2006 added
- BIT fields of MySQL 5.0 and above are now represented as TLargeintField
- FastReport 3.20 support added
- Added server version checking on Prepare method
- Added capability to close DataSet quicker when FetchAll property is False
- Improved performance of updating recordsets with multiple fields
- TCustomDADataSet.Locate now centers position in DBGrid
- Added support for MIDAS TDataSet.PSExecuteStatement ResultSet parameter

07-Dec-05 New Features in MySQL Data Access Components 4.00.2:
- Added 'delimiter' keyword support in TMyScript
- TCustomDADataSet.FindKey, TCustomDADataSet.FindNearest methods added for BDE compatibility
- Added BIT and INTEGER types support in parameters of stored procedures

02-Sep-05 New Features in MySQL Data Access Components 4.00.1:
- Deferred detail dataset refresh feature with TCustomDADataSet.Options.DetailDelay property added
- TCustomMyConnection.Ping behavior in case connection loss improved
- FieldDefs.Update behavior in case of temporary tables improved
- Added capability to prevent simultaneous access of several MyEmbConnection instances to single data folder

29-Jul-05 New Features in MySQL Data Access Components 4.00:
- Unicode support added
- Enhanced support for Embedded MySQL Server with TMyEmbConnection component added
- Binary protocol support for MySQL 4.1 and above added
- Encrypted SSL connections support with OpenSSL added
- Enhanced support for national charsets added with TMyConnectionOptions.Charset property
- BLOB compression support added
- RefreshQuick for TCustomMyDataSet added
- Retrieve field's default value added with TCustomMyDataSet.Options.DefaultValues property
- Large amount of data support for TMyDump added
- Server start/stop ability for TMyServerControl added
- TMyBuilder component added for easy using SQL Builder for MySQL at run-time
- Optimized macros processing
- FAQ added
- Tested with MySQL server 5.0.9

30-May-05 New Features in MySQL Data Access Components 3.55:
- MySQL 5.0.3 BIT type support added
- Optimized MySQLMonitor BLOB parameters processing
- Ability of automatic preparing query with TCustomDADataSet.Options.AutoPrepare property added
- Ability to synchronize position at different DataSets with TCustomDADataSet.GotoCurrent method added

21-Jan-05 MySQL Data Access Components 3.50:
- Support for Delphi 2005 added
- TMySQLMonitor.OnSQL can return statement encoded to an escaped SQL string
- Support for ConnectionTimeout in TMyConnection.ExecSQL added
- CommandTimeout default value set to 0 (infinite)
- TCustomDADataSet.UpdateObject support for MIDAS added
- Lock Demo added
- DECIMAL column type in MySQL 5.0.3 support added
- Update Pack 3 is required for Delphi 8
21-Oct-04 New Features in MySQL Data Access Components 3.30:
- Full support for all current authentication protocols added
- Generating update SQL for tables from other database added
- TCustomMyDataSet.Options.EnableBoolean property added
- TMyConnection.ThreadId property added
- lxPartialCompare option for DataSet.LocateEx added
- FastReport3 engine and demo added
- Ability to store only a part of data in TMyDump.BackupQuery added
- Creating additional connection for TMyDump disabled
- TCustomMyDataSet.CommandTimeout property added
- "True" value for boolean fields and parameters stored as "1"

10-Sep-04 New Features in MySQL Data Access Components 3.10.2:
- Common class DADataAdapter isolated to Devart.Dac.AdoNet.dll library

22-Jul-04 New Features in MySQL Data Access Components 3.10.1 new features:
- Fatal errors processing improved
- TINYINT(1) fields now represented as TBooleanField

08-Jul-04 New Features in MySQL Data Access Components 3.10:
- Local sorting ability with TMemDataSet.IndexFieldNames added
- TCustomMyTable.IndexDefs property added
- TMyConnection.Options.NumericType property added
- TMyStoredProc component added
- MyDataAdapter component added

29-Apr-04 New Features in MySQL Data Access Components 3.00.1:
- TCustomMyDataSet.Options.LongStrings property added
What's New

- TMyLoader.OnPutData event published again
- Trial version IDE warning disabled
- TCRColumn.TotalValue property added

09-Apr-04 New Features in MySQL Data Access Components 3.00:
- Support for Delphi 8 added
- Connection pooling support
- Performance improved
- TMyLoader performance greatly improved
- TCRGrid sources in Standard edition
- .NET Windows Forms demo project added
- ASP.NET demo project added
- Global variable MySQLClientLibrary added
- New time trial limitation

05-Feb-04 New Features in MySQL Data Access Components 2.00.3:
- SELECT '' support added
- Method TMyConnection.Ping added
- Method TMyConnection.ExecuteInfo added
- Mouse wheel support added to CRDBGrid
- Embedded MySQL Server Demo added
- ConnectDialog Demo added

30-Dec-03 New Features in MySQL Data Access Components 2.00.2:
- BDE Migration Wizard algorithm optimized
- Limited MySQL server 4.1.1 support added
- If libmysql.dll not found then raise EOSError (instead of Exception)
- Property TCustomMyDataSet.InsertId: int64 added
- Timestamp support added for CheckRowVersion = True

24-Nov-03 New Features in MySQL Data Access Components 2.00.1:
• Property MyConnection.Options.Direct is set to True by default
• TCustomMyDataSet.Lock method added
• AutoInc fields can be modified now

02-Oct-03 New Features in MySQL Data Access Components 2.00:
• Access to MySQL without client library using DirectMySQLObjects by Cristian Nicola
• Prepare support and new parameter binding schema for MySQL 4.1 added
• Supports working with MySQL server and Embedded server at the same time
• BDE migration wizard
• TMyDump component to store a database or its parts as a script
• TMyBackup component for backup coping specified tables on the server
• TMyServerControl component to manage the server and standard service tasks execution
• TMyLoader component for fast loading data to the server
• New options of TMyConnection such as Compress, Protocol, Direct and Embedded added
• New properties ClientVersion, ServerVersion were added to TMyConnection
• Method ExecSQL in TMyConnection added
• Methods GetTableNames and GetDatabaseNames in TMyConnection added
• Property TMyConnection.Charset added
• Property TMyConnection.IsolationLevel added
• Methods LockTable and UnlockTable added to TCustomMyDataset
• Properties Limit and Offset added to TCustomMyTable
• Method TCustomMyTable.EmptyTable added
• FetchAll set to True by default
• Large SQL (INSERT/UPDATE BLOB’s) executing performance greatly improved

06-Jun-03 New Features in MySQL Data Access Components 1.50:
• Embedded MySQL Server support added
• MySQL Server 4.1 limited support added
• Properties Port and Database in ConnectForm added
• RefreshRecord performance improved
• InfoPower demos added
• 'Explain query...' added to design-time MyQuery menu
• 'Show CREATE...' added to design-time MyQuery and MyTable menus
• SQL Generator improved - support for complicated statements added
• SQL Generator improved - "Quote names" checkbox added
• Complex keys support added
• Design-time SQL Generator was simplified
• TParam -> TDAParam
• Embedded MySQL Server support added for Kylix
• Check for datadir present added
• Changed behavior on calculating affected rows count

04-Apr-03 New Features in MySQL Data Access Components 1.30.2:
• Unit MySQLAccess renamed to MyClasses
• Property TMyDataSetOptions.LongStrings removed
• Parameters parsing improved. Symbol ':' in string literals is ignored
• Search algorithm for 'libmysqlclient.so' under Linux improved

24-Feb-03 New Features in MySQL Data Access Components 1.30.1:
• Refresh improved - current record is restored after Refresh call
• Property MyConnection.Options.KeepDesignConnected added
• Property MyConnectDialog.StoreLogInfo published
• Property MyScript.DataSet was published
• Property TMyCommand.InsertId: int64 added
• TINYTEXT -> TMemoField, TINYBLOB -> TBlobField
• Support for TIMESTAMP (10), TIMESTAMP (4), TIMESTAMP (2) added
• Support for LIKE expressions in Filter property added (D2706)

30-Jan-03 New Features in MySQL Data Access Components 1.30:
• MySQL v4.0 support added
• Dataset 'with many fields' update performance improved
• Improved performance for opening queries with lot of parameters

26-Dec-02 New Features in MySQL Data Access Components 1.20:
• Kylix2 and Kylix3 support
• ReportBuilder demos added
• DBMonitor client implementation moved to COM server
• Fetch performance improved for DataSet.FetchAll = True
• ‘Connection Lost’ error processing improved

08-Oct-02 New Features in MySQL Data Access Components 1.10:
• Delphi 7 support
• New memory management model for ftString and ftVarBytes types. Allows significantly decrease memory usage on large tables fetch. Controlled by FlatBuffers dataset option
• Support for blob fields in CachedUpdates mode

2 General Information

This section contains general information about MySQL Data Access Components
• Overview
• Features
• Requirements
• Compatibility
• Using Several DAC Products in One IDE
• Component List
• Hierarchy Chart
• Editions
• Licensing and Subscriptions
• Getting Support
• Frequently Asked Questions

MySQL Data Access Components (MyDAC) is a library of components that provides direct
access to MySQL database servers from Delphi, C++Builder, and Lazarus (Free Pascal). MyDAC can connect directly to MySQL server or work through the MySQL client library. The MyDAC library is designed to help programmers develop faster and cleaner MySQL database applications. MyDAC is a complete replacement for standard MySQL connectivity solutions and presents an efficient alternative to the Borland Database Engine for access to MySQL.

The MyDAC library is actively developed and supported by the Devart Team. If you have any questions about MyDAC, email the developers at mydac@devart.com or visit MyDAC online at https://www.devart.com/mydac/.

**Advantages of MyDAC Technology**

MyDAC is a direct database connectivity wrapper built specifically for the MySQL server. MyDAC offers wide coverage of the MySQL feature set, supports both client and direct connection modes, and emphasizes optimized data access strategies.

**Wide Coverage of MySQL Features**

By providing access to the most advanced database functionality, MyDAC allows developers to harness the full capabilities of the MySQL server and optimize their database applications. MyDAC provides a complete support of MySQL Embedded Server, row-level locking, using HANDLER statements, MySQL administration tasks. View the full list of supported MySQL features in [Features](#).

**Native Connection Options**

MyDAC offers two connection modes to the MySQL server: Direct connection and connection through the standard MySQL Client in Client mode. MyDAC-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. See the [How does MyDAC work](#) section.

**Optimized Code**

The goal of MyDAC is to enable developers to write efficient and flexible database applications. The MyDAC library is implemented using optimized code and advanced data access algorithms. Component interfaces undergo comprehensive performance tests and are designed to help you write thin and efficient product data access layers. Find out more about how to use MyDAC to optimize your database applications in [Increasing Performance](#).

**Compatibility with other Connectivity Methods**
The MyDAC interface retains compatibility with standard VCL data access components BDE. Existing BDE-based applications can be easily migrated to MyDAC and enhanced to take advantage of MySQL-specific features. Migration of a Delphi project can be automated with the BDE Migration Wizard. Find out more about Migration Wizard in Migration from BDE.

Development and Support

MyDAC is a MySQL connectivity solution that is actively developed and supported. MyDAC comes with full documentation, demo projects, and fast (usually within one business day) technical support by the MyDAC development team. Find out more about how to get help or submit feedback and suggestions to the MyDAC Development Team in Getting Support.

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

Key Features

- **Direct** access to server data without using client library. Does not require installation of other data provider layers (such as BDE and ODBC)
- VCL, LCL and FMX versions of library available
- Full support of the latest versions of MySQL Server
- Support for all MySQL 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 TMyQuery, TMyTable, and TMyStoredProc components
- **Unicode** and national charset support
- Supports many MySQL-specific features, such as locking, SET and ENUM types
- Advanced script execution functionality with TMyScript component
- Support for using macros in SQL
- Easy migration from BDE with Migration Wizard
- Lets you use Professional Edition of Delphi and C++Builder to develop client/server applications
- Included annual MyDAC Subscription with Priority Support
- Licensed royalty-free per developer, per team, or per site

The full list of MyDAC features is available in Features.

How does MyDAC work?

MyDAC allows you to connect to MySQL in two ways: in the Direct mode over TCP/IP or in
the Client mode using MySQL client software. MyDAC works directly through the native MySQL interface to avoid using BDE and ODBC. The Direct mode can be enabled or disabled using the Direct option.

In the Direct mode, MyDAC connects to MySQL directly, without using the MySQL client software.

In the Client mode, MyDAC connects to MySQL using the MySQL client library. The MySQL client library is supplied with MySQL server.

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

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 without using client library. 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:
- Disconnected Model with automatic connection control for working with data offline
- Local Failover for detecting connection loss and implicitly reexecuting certain operations
- Support for all existing MySQL protocols including the prepared statement (binary) protocol
- Secure connections with SSL, SSH, and HTTP/HTTPS tunneling (using SecureBridge)
- Full support for all current authentication protocols
- Ability to search for installed MySQL servers in a local network
- Connection timeout and command timeout management

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

MySQL Server technology support:
• Fast record insertion with TMyLoader component
• HANDLER syntax support
• Transaction isolation level support
• Possibility to retrieve last auto-incremented value
• Session identifier retrieval
• Server object information retrieval
• Row-level and table-level locking support

Performance:
• High overall performance
• Fast controlled fetch of large data blocks
• Optimized string data storing
• Advanced connection pooling
• High performance applying of 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 TMyQuery, TMyTable and TMyStoredProc components
• Automatic record refreshing
• 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 **TMyTable** component
- **BLOB compression** support
- Support for using macros in SQL
- **FmtBCD fields support**
- Ability to customize update commands by attaching external components to **TMyUpdateSQL** objects
- Ability to perform MySQL administration tasks with the **TMyServerControl** component
- **Value range retrieval for ENUM and SET fields**
- 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 **TMyDump**, **TMyBackup** components

Script execution:
- Advanced script execution features with **TMyScript** component
- Support for executing **individual statements** in scripts
- Support for **executing huge scripts stored in files** with dynamic loading
- **Optimized multi-statement script execution**
- Ability to use standard MySQL client tool syntax in scripts
- Ability to break long-running query execution

SQL execution monitoring:
- Extended SQL tracing capabilities provided by **TMySQLMonitor** 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
- Easy migration from BDE with Migration Wizard
- More convenient data source setup with the TMyDataSource 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 MyDAC Subscription with Priority Support
- Licensed royalty-free per developer, per team, or per site

2.3 Requirements

Requirements for using MyDAC in Direct mode

If you use MyDAC to connect to MySQL in Direct mode, you do not need to have MySQL client library on your machine or deploy it with your MyDAC-based application.

Requirements for using MyDAC in Client mode

If you use MyDAC to connect to MySQL in Client mode, you need to have access to the MySQL client library. In particular, you will need to make sure that the MySQL client library is installed on the machines your MyDAC-based application is deployed to. MySQL client library is libmysql.dll file for Windows, or libmysqlclient.so (libmysqlclient.so.X) for Linux. Please
refer to descriptions of LoadLibrary() and dlopen() functions accordingly for detailed information about MySQL client library file location. You may need to deploy the MySQL client library with your application or require that users have it installed.

Requirements for using Embedded MySQL server

If you are working with Embedded server, you should have access to Embedded MySQL server library (libmysqld.dll). For more information visit Using Embedded server.

MySQL Compatibility

MyDAC supports the following versions of MySQL:

- MySQL server: 8.0, 6.0, 5.7, 5.6, 5.5, 5.1, 5.0, 4.1, 4.0, 3.23.
- MySQL embedded server: 8.0, 6.0, 5.7, 5.6, 5.5, 5.1, 4.1, 4.0.

MariaDB Compatibility

MariaDB server: 5.x-10.x.

Microsoft Azure Database for MySQL Compatibility

MyDAC supports Microsoft Azure Database for MySQL.

Amazon RDS for MySQL Compatibility

MyDAC supports Amazon RDS for MySQL and Amazon Aurora.

Google Cloud for MySQL Compatibility

MyDAC supports Google Cloud for MySQL.

IDE Compatibility

MyDAC 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 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 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 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 iOS
  - Embarcadero Delphi 10.1 Berlin for Android
  - Embarcadero C++Builder 10.1 Berlin for Windows
  - Embarcadero C++Builder 10.1 Berlin for macOS
  - Embarcadero C++Builder 10.1 Berlin for iOS
  - Embarcadero C++Builder 10.1 Berlin for Android

- 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
  - Embarcadero C++Builder XE6 for iOS
  - Embarcadero C++Builder XE6 for Android

- Embarcadero RAD Studio XE5 (Requires Update 2)
  - Embarcadero Delphi XE5 for Windows
  - Embarcadero Delphi XE5 for macOS
  - Embarcadero Delphi XE5 for iOS

© 2021 Devart
- Embarcadero Delphi XE5 for Android
- Embarcadero C++Builder XE5 for Windows
- Embarcadero C++Builder XE5 for macOS
- Embarcadero C++Builder XE5 for iOS
- Embarcadero RAD Studio XE4
  - Embarcadero Delphi XE4 for Windows
  - Embarcadero Delphi XE4 for macOS
  - Embarcadero Delphi XE4 for iOS
  - Embarcadero C++Builder XE4 for Windows
  - Embarcadero C++Builder XE4 for macOS
- Embarcadero RAD Studio XE3 (Requires Update 2)
  - Embarcadero Delphi XE3 for Windows
  - Embarcadero Delphi XE3 for macOS
  - Embarcadero C++Builder XE3 for Windows
  - Embarcadero C++Builder XE3 for macOS
- Embarcadero RAD Studio XE2 (Requires Update 4 Hotfix 1)
  - Embarcadero Delphi XE2 for Windows
  - Embarcadero Delphi XE2 for macOS
  - Embarcadero C++Builder XE2 for Windows
  - Embarcadero C++Builder XE2 for macOS
- Embarcadero RAD Studio XE
  - Embarcadero Delphi XE
  - Embarcadero C++Builder XE
- Embarcadero RAD Studio 2010
  - Embarcadero Delphi 2010
  - Embarcadero C++Builder 2010
- CodeGear RAD Studio 2009 (Requires Update 3)
  - CodeGear Delphi 2009
  - 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.
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.

2.6 Component List

This topic presents a brief description of the components included in the MySQL Data Access Components library. Click on the name of each component for more information. These components are added to the MyDAC 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 MyDAC components are included in all MyDAC editions. MyDAC Professional and Developer Edition components are not included in MyDAC Standard Edition.

Basic MyDAC components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyConnection</td>
<td>Lets you set up and control connections to MySQL database server.</td>
</tr>
<tr>
<td>TMyQuery</td>
<td>Uses SQL statements to retrieve data from MySQL table or tables and supply it to one or more data-aware components through a TDataSource component. Provides flexible data update functionality.</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TMyCommand</td>
<td>Executes SQL statements and stored procedures, which do not return rowsets.</td>
</tr>
<tr>
<td>TMyTable</td>
<td>Lets you retrieve and update data in a single table without writing SQL statements.</td>
</tr>
<tr>
<td>TMyStoredProc</td>
<td>Executes stored procedures and functions.</td>
</tr>
<tr>
<td>TMyUpdateSQL</td>
<td>Lets you tune update operations for a DataSet component.</td>
</tr>
<tr>
<td>TMyDataSource</td>
<td>Provides an interface between MyDAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td>TMyScript</td>
<td>Executes sequences of SQL statements.</td>
</tr>
<tr>
<td>TMySQLMonitor</td>
<td>Interface for monitoring dynamic SQL execution in MyDAC-based applications.</td>
</tr>
<tr>
<td>TMyConnectDialog</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>

**MyDAC Professional Edition components**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyEncryptor</td>
<td>Represents data encryption and decryption in client application.</td>
</tr>
<tr>
<td>TMyLoader</td>
<td>Provides quick loading data to MySQL database.</td>
</tr>
<tr>
<td>TMyDump</td>
<td>Serves to store a database or its parts as a script and also to restore database from received script.</td>
</tr>
<tr>
<td>TMyBackup</td>
<td>Serves for backup copying specified tables on the server.</td>
</tr>
<tr>
<td>TMyServerControl</td>
<td>Serves to control the server and execution of standard service tasks.</td>
</tr>
</tbody>
</table>
**TMyEmbConnection**  
Is used to establish connection to Embedded MySQL server.

**TMyMetaData**  
Retrieves metadata on specified SQL object.

**TCRBatchMove**  
Transfers data between all types of TDataSets descendants. This component is placed on the Data Access page of the Component palette.

## See Also
- [Hierarchy chart](#)

© 1997-2021 Devart. All Rights Reserved.

### 2.7 Hierarchy Chart

Many MyDAC classes are inherited from standard VCL/LCL classes. The inheritance hierarchy chart for MyDAC is shown below. The MyDAC 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
           |---TCustomMyConnection
           |   |---TMyConnection
           |   |---TMyEmbConnection
       |---TDataSet
           |---TMemDataSet
           |   |---TCustomDADataset
           |     |---TCustomMyDataSet
           |     |   |---TMyQuery
           |     |   |---TCustomMyTable
           |     |   |---TMyTable
```
2.8 Editions

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

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

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

You can get **Source Access** to the implementation of all the component classes in MyDAC by purchasing a special MyDAC Professional Edition with Source Code. The source code of DataSet Manager and Migration Wizard is not distributed.

The matrix below compares the features of MyDAC editions. See [Features](#) for the detailed list of MyDAC features.

### MyDAC Edition Matrix

<table>
<thead>
<tr>
<th>Feature</th>
<th>Standard</th>
<th>Professional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Direct connectivity</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Connection without MySQL client library</strong></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Desktop Application Development</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>macOS</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Linux</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Mobile Application Development</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iOS</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Android</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Data Access Components</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base Components:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMyConnection</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TMyQuery</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TMyCommand</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TMyTable</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
### MySQL Data Access Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Available</th>
<th>Supported</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Script executing</strong></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>TMyScript</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Fast data loading into the server</strong></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>TMyLoader</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

#### MySQL Specific Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Available</th>
<th>Supported</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtaining metadata about database objects</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>TMyMetaData</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Connection with Embedded MySQL Server</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>TMyEmbConnection</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>MySQL server control</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>TMyServerControl</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>SQL Builder managing for MySQL Add-in</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>TMyBuilder</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Storing a database as a script</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>TMyDump</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Backup Service</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>TMyBackup</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

#### DataBase Activity Monitoring

<table>
<thead>
<tr>
<th>Component</th>
<th>Available</th>
<th>Supported</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitoring of per-component SQL execution</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>TMySQLMonitor</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

#### Additional components

<table>
<thead>
<tr>
<th>Component</th>
<th>Available</th>
<th>Supported</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced connection dialog</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>TMyConnectDialog</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Data encryption and decryption</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>TMyEncryptor</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Data storing in memory table</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>TVirtualTable</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Dataset that wraps arbitrary non-tabular data</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>TVirtualDataSet</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Advanced DBGrid with extended functionality</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>TCRDBGrid</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Records transferring between datasets</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>TCRBatchMove</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Design-Time Features

<table>
<thead>
<tr>
<th>Component</th>
<th>Available</th>
<th>Supported</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enhanced component and property editors</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
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

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 MyDAC 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:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Professional Edition</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migration Wizard</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DataSet Manager</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Cross IDE Support</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lazarus and Free Pascal Support</td>
<td>×</td>
<td>SRC¹</td>
</tr>
</tbody>
</table>

1 Available only in Professional Edition with Source Code.
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 MyDAC 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 MyDAC. You can distribute MyDAC 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 from any copy of the Software or any copy of the written materials, accompanying the Software.

This Agreement contains the total agreement between the two parties and supersedes any other agreements, written, oral, expressed, or implied.

© 1997-2021 Devart. All Rights Reserved.

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

Support Options
There are a number of resources for finding help on installing and using MyDAC.
- You can find out more about MyDAC installation or licensing by consulting the Licensing and
FAQ sections.
- You can get community assistance and technical support on the MyDAC Community Forum.
- You can get advanced technical assistance by MyDAC developers through the MyDAC Priority Support program.

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

MyDAC Priority Support

MyDAC Priority Support is an advanced product support service for getting expedited individual assistance with MyDAC-related questions from the MyDAC 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 MyDAC Subscription.

To get help through the MyDAC Priority Support program, please send an email to mydac@devart.com describing the problem you are having. Make sure to include the following information in your message:
- The version of Delphi, C++Builder you are using.
- Your MyDAC Registration number.
- Full MyDAC edition name and version number. You can find both of these in the About sheet of TMyConnection Editor or from the MySQL | About menu.
- Versions of the MySQL server and client you are using.
- A detailed problem description.
- If possible, a small test project that reproduces the problem. Please include definitions for all database objects and avoid using third-party components.

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback

2.11 Frequently Asked Questions

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

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

Installation and Deployment

© 2021 Devart
1. I am having a problem installing MyDAC or compiling MyDAC-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 MyDAC 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 MyDAC, 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 MyDAC-based applications to work?

Usually, you do not need any additional files. The only exceptions to this rule are listed below:
- If you are using MySQL Embedded server (if you are using TMyConnection with TMyConnection.Options.Embedded = True or TMyEmbConnection), you need the server itself (libmysqld.dll) and the service files for it, for example errmsg.sys.
- If you are connecting in Client mode, (TMyConnection.Options.Direct = False), you need libmysql.dll.
- If you are using SSL (TMyConnection.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 "MySQL client library couldn't be loaded."

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

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

2. Can I create components using MyDAC?

You can create your own components that are inherited from MyDAC or that use the MyDAC 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. What licensing changes can I expect with MyDAC 5.00?

The basic MyDAC license agreement will remain the same. With MyDAC 5.00, the MyDAC Edition Matrix will be reorganized and a new MyDAC Subscription Program will be introduced.

4. What do the MyDAC 5.00 Edition Levels correspond to?

MyDAC 5.00 will come in six editions: Trial, Standard, Professional, Professional with Sources, Developer, and Developer with Sources.

When you upgrade to the new version, your edition level will be automatically updated using the following Edition Correspondence Table.

### Edition Correspondence Table for Upgrading to MyDAC 5.00

<table>
<thead>
<tr>
<th>Old Edition Level</th>
<th>New Edition Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>- No Correspondence -</td>
<td>MyDAC Standard Edition</td>
</tr>
<tr>
<td>MyDAC Professional Edition</td>
<td>MyDAC Professional Edition with Sources</td>
</tr>
<tr>
<td>- No Correspondence -</td>
<td>MyDAC Developer Edition</td>
</tr>
<tr>
<td>- No Correspondence -</td>
<td>MyDAC Developer Edition with Sources</td>
</tr>
<tr>
<td>MyDAC Trial Edition</td>
<td>MyDAC Trial Edition</td>
</tr>
</tbody>
</table>
The feature list for each edition can be found in the MyDAC documentation and on the MyDAC website.

5. I have a registered version of MyDAC. Will I need to pay to upgrade to future versions?

After MyDAC 5.00, all upgrades to future versions are free to users with an active MyDAC Subscription.

Users that have a registration for versions of MyDAC prior to MyDAC 5.00 will have to first upgrade to MyDAC 5.00 to jump in on the Subscription program.

6. What are the benefits of the MyDAC Subscription Program?

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

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

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

The MyDAC Subscription Program is available for registered users of MyDAC 5.00 and higher.

7. Can I use my version of MyDAC after my Subscription expires?

Yes, you can. MyDAC version licenses are perpetual.

8. I want a MyDAC Subscription! How can I get one?

An annual MyDAC Subscription is included when ordering or upgrading to any registered (non-Trial) edition of MyDAC 5.00 or higher.

You can renew your MyDAC Subscription on the MyDAC Ordering Page. For more information, please contact sales@devart.com.

9. Does this mean that if I upgrade to MyDAC 5 from MyDAC 4, I’ll get an annual
MyDAC Subscription for free?

Yes.

10. How do I upgrade to MyDAC 5.00?

To upgrade to MyDAC 5.00, you can get a Version Update from the MyDAC Ordering Page. For more information, please contact sales@devart.com.

Performance

1. How productive is MyDAC?

MyDAC uses a low-level protocol to access the database server. This allows MyDAC to achieve high performance. From time to time we compare MyDAC with other products, and MyDAC 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, MyDAC 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 enable syntax highlighting in MyDAC component editors at design time?

Download and install dbForge Fusion for MySQL. In addition to syntax highlighting, MySQL Developer Tools provides a lot of additional features. Alternatively, you can download and install the freeware SynEdit component set.

2. How can I quickly convert a project from BDE to MyDAC?

To quickly migrate your project from BDE you can use the BDE Migration Wizard. To start it, open your project and choose BDE Migration Wizard from the MyDAC menu of your IDE.

3. How can I determine which version of MyDAC I am using?

You can determine your MyDAC version number in several ways:

- During installation of MyDAC, consult the MyDAC Installer screen.
- After installation, see the history.html file in your MyDAC installation directory.
- At design-time, select MySQL | About MyDAC from the main menu of your IDE.
4. **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.

5. **How can I execute a query saved in the SQLInsert, SQLUpdate, SQLDelete, or SQLRefresh properties of a MyDAC 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 MyDAC 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 MyDAC 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 MyDAC dataset object will generate the query itself using the appropriate insert, update, delete, or refresh record syntax.

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

Use the `TMyConnection.GetDatabaseNames` method.

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

Use the `TMyConnection.GetTableNames` method.

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

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

**General Questions**

1. **I would like to develop an application that works with MySQL Server. Should I use MyDAC or DbxMda?**

   DbxMda is our dbExpress driver for MySQL. dbExpress technology serves for providing a more or less uniform way to access different servers (SQL Server, MySQL, 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.

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

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

   MyDAC 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 MySQL, it is probably better to use DbxMda. In other cases, especially when migrating from BDE or ADO, you should use MyDAC.

2. **Are the MyDAC connection components thread-safe?**

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

3. **Behaviour of my application has changed when I upgraded MyDAC. How can I restore the old behaviour with the new version?**
We always try to keep MyDAC compatible with previous versions, but sometimes we have to change behaviour of MyDAC 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 MyDAC help. This topic describes such changes, and how to revert to the old MyDAC 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 TCustomMyDataSet.Options.StrictUpdate topic in MyDAC 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 'MySQL server has gone away' or 'Lost connection to MySQL 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 TMyConnection.ConnectionTimeout or TCustomMyDataSet.CommandTimeout is too small. To check this hypothesis, try setting TCustomMyDataSet.CommandTimeout to 0 (infinitive) and TMyConnection.ConnectionTimeout to 300.
- Server side: MySQL server has closed the connection. You can read a detailed description of all possible reasons for this to happen in the MySQL Reference Manual. 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 MyConnection.Ping with an interval less than wait_timeout. Use TTimer in TMyConnection thread to accomplish this task.

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

If the connection is lost, MyDAC tries to reconnect to server. However, your last command will probably not be executed, and you should repeat it again. MyDAC 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 proprety and the CreateConnection option in MyDAC help for more information.

Another alternative that prevents the application from hanging is to switch to the InnoDB storage engine from MyISAM (FetchAll stays False). An application may hang because MyISAM tables can get locked in a read/write collision. If you try to update a table that is not fetched out, MySQL blocks the thread and waits untill the table is completely fetched. For details please refer to the MySQL Reference Manual, the Locking Issues section.

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

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

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

MyStoredProc.Prepare;
MyStoredProc.Open;
MyStoredProc.UnPrepare;
3 Getting Started

This section introduces MySQL Data Access Components. It contains the information on how to install MySQL Data Access Components, quick walktroughs to get started developing applications with it, information on technical licensing and deployment, and brief description of MyDAC documentation and samples.

- Installation
- Migration Wizard
- Migration from BDE
- Connecting to MySQL
- Connecting to MySQL Embedded
- Creating Database Objects
- Deleting Data From Tables
- Inserting Data Into Tables
- Retrieving Data
- Modifying Data
- Stored Procedures
- Working With Result Sets Using Stored Procedures

3.1 Installation

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

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

Two versions of MyDAC 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 MyDAC may be incompatible with older versions of ODAC, IBDAC, and SDAC.

So before installing a new version of MyDAC, uninstall any previous version of MyDAC 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 mydac@devart.com

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

### Installed packages

**Note:** %MyDAC% denotes the path to your MyDAC installation directory.

**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>mydacXX.bpl</td>
<td>MyDAC run-time package</td>
<td>Windows\System32</td>
</tr>
<tr>
<td>dclmydacXX.bpl</td>
<td>MyDAC design-time package</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>dclmysqlmonXX.bpl</td>
<td>TMySQLMonitor component</td>
<td>Delphi\Bin</td>
</tr>
<tr>
<td>mydacvclXX.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>
* Not included in Delphi 5. In this IDE this functionality is distributed among the other packages.

### Additional packages for using MyDAC 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>
<tr>
<td>mymigwizardXX.dll</td>
<td>MyDAC BDE Migration wizard</td>
<td>%MyDAC%\Bin</td>
</tr>
</tbody>
</table>

### Environment Changes

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

For all instructions, replace `%MyDAC%` with the path to your MyDAC installation directory.

**Delphi**

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

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

**C++Builder**

**C++Builder 6:**

- `$\{BCB\} \ MyDAC\ Lib` should be included in the Library Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.
- `$\{BCB\} \ MyDAC\ Include` should be included in the Include Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.

**C++Builder 2006, 2007:**

- `$\{BCB\} \ MyDAC\ Lib` should be included in the Library search path of the Default Project Options accessible from Project | Default Options | C++Builder | Linker | Paths and Defines.
- `$\{BCB\} \ MyDAC\ 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 MyDAC Installer performs C++Builder environment changes automatically for compiled versions of MyDAC.
versions of MyDAC.

**Lazarus**

The MyDAC installation program only copies MyDAC files. You need to install MyDAC packages to Lazarus IDE manually. Open `%MyDAC%\Source\Lazarus1\dclmydac10.lpk` (for Trial version `%MyDAC%\Packages\dclmydac10.lpk`) file in Lazarus and press the Install button. After that Lazarus IDE will be rebuilt with MyDAC packages.

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

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

**Installation of Additional Components and Add-ins**

**dbForge Fusion for MySQL**

dbForge Fusion for MySQL is a powerful database development and administration tool for MySQL. dbForge Fusion for MySQL is available as an add-in for Delphi and C++Builder 2009, CodeGear RAD Studio 2007, or as a standalone application. For more information, visit the [dbForge Fusion for MySQL page online](#).

**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 MyDAC. DBMonitor is intended to hamper application being monitored as little as possible. For more information, visit the [DBMonitor page online](#).

© 1997-2021 Devart. All Rights Reserved.

© 2021 Devart

3.2 **Migration Wizard**

*Note: Migration Wizard is only available for Delphi.*

BDE Migration Wizard allows you to convert your BDE projects to MyDAC. This wizard replaces BDE components in a specified project (.dfm and .pas files) with MyDAC components.
To convert a project, perform the following steps.

- Select **BDE Migration Wizard** from the **MyDAC** menu
- Select **Replace BDE components** to replace corresponding components with MyDAC and press the Next button.
- Select the location of the files to search - current open project or disc folder.
- If you have selected Disc folder on the previous step, specify the required folder and specify whether to process subfolders. Press the Next button.
- Select whether to make backup (it is highly recommended to make a backup), backup location, and log parameters, and press the Next button. Default backup location is RBackup folder in your project folder.
- Check your settings and press the Finish button to start the conversion operation.
- The project should be saved before conversion. You will be asked before saving it. Click Yes to continue project conversion.

After the project conversion it will be reopened.

The Wizard just replaces all standard BDE components. Probably you will need to make some changes manually to compile your application successfully.

If some problems occur while making changes, you can restore your project from backup file. To do this perform the following steps.

- Select **BDE Migration Wizard** from the **MyDAC** menu
- Select Restore original files from backup and press the Next button.
- Select the backup file. By default it is RExpert.reu file in RBackup folder of your converted project. Press the Next button.
- Check your settings and press the Finish button to start the conversion operation.
- Press Yes in the dialog that appeared.

Your project will be restored to its previous state.

**See Also**
- [Migration from BDE](#)

© 1997-2021
Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback

### 3.3 Migration from BDE

In MyDAC the interests of BDE application developers were taken into consideration. So
starting to use MyDAC after working with BDE would be easy even for developing complex projects. Moreover, MyDAC does not have problems like ones with LiveQuery and compatibility of applications developed using different versions in BDE.

Abandoning BDE gives one more important advantage - positive effect on performance. Instead of complex BDE-ODBC drivers system it uses the fastest access - directly to MySQL server. Also access to MySQL Embedded server is supported.

MyDAC provides special Wizard to simplify the conversion of already existing projects. This Wizard replaces BDE-components in the specified project (dfm-and pas-files) to MyDAC. BDE-components that will be replaced:

- TDatabase -> TMyConnection
- TQuery -> TMyQuery
- TTable -> TMyTable
- TUpdateSQL -> TMyUpdateSQL

To run the Wizard, select BDE Migration Wizard item in MyDAC menu and follow the instructions. BDE Migration Wizard does not support C++Builder.

**Note**: The Wizard serves only to simplify routine operations and after the conversion project might be uncompiled.

Below is the list of properties and methods which cannot be converted automatically. Here you can find hints for users to simplify manual replacement.

**TDatabase**

- **AliasName** - specific BDE property. Not supported by MyDAC
- **DatabaseName** - has a different meaning in BDE and MyDAC. At MyDAC it means MySQL Server database. See TMyConnection.Database for details
- **Locale** - see TMyConnection.Options CharSet
- **KeepConnection** - not supported by MyDAC
- **Params** - see TMyConnection properties
- **Session, SessionAlias, SessionName** - MyDAC does not need global management of a group of database connections in an application. So these properties are not supported
- **Temporary** - has no sense in MyDAC. Additional connections are created but not available for the user. See TCustomMyDataSet.FetchAll = False for details
- **TraceFlags** - see TCustomDASQLMonitor.TraceFlags
- **TransIsolation** - see IsolationLevel
- **Execute** - use ExecSQL instead of this method
• FlushSchemaCache - not supported by MyDAC
• GetFieldNames - not supported by MyDAC
• IsSQLBased - not supported by MyDAC. For MySQL must be always True
• ApplyUpdates - parameters are not supported. To update only specified DataSets use `TMemDataset.ApplyUpdates`. Update is performed within a transaction.

**TBDEDataSet**
• BlockReadSize - see `TCustomDADataset.FetchRows`
• CacheBlobs - MySQL Server does not provide service of suspended BLOB loading
• KeySize - specific BDE property. Not supported by MyDAC.

**TDBDataSet**
• AutoRefresh - supported through `TCustomDADataset.RefreshOptions`
• DBFlags, DBHandle, DBLocate, DBSession, Handle - BDE-specific property. Not supported by MyDAC
• SessionName - not supported by MyDAC
• UpdateMode - not supported by MyDAC. By default, the behaviour corresponds `upWhereKeyOnly`. To change this behaviour see `TCustomDADataset.SQLUpdate`, `TCustomDADataset.SQLDelete`, `TCustomDADataset.SQLRefresh`, and `TCustomMyDataSet.Options.CheckRowVersion`.

**TQuery**
• Constrained - specific BDE property. Not supported by MyDAC
• DataSource - see `TCustomDADataset.MasterSource`
• Local - specific BDE property. Not supported by MyDAC
• RequestLive - almost all query result sets are updatable. See `TMyQuery.UpdatingTable`, `TCustomDADataset.ReadOnly`, `CanModify`, `TCustomDADataset.SQLInsert`, `TCustomDADataset.SQLUpdate`, `TCustomDADataset.SQLDelete`.
• Text - specific BDE property. Not supported by MyDAC.

**TTable**
• DefaultIndex - not used in MyDAC. If you need to sort a table by any field see `P:\Devart.MyDac.TCustomMyTable.OrderFields`, `TMemDataSet.IndexFieldNames`
• Exists, CreateTable, AddIndex, DeleteIndex, StoreDefs, Deletetable, TableType - MyDAC does not allow to create tables by using TTable. If you need to create a table execute 'CREATE TABLE ...' query or use any special third-party tools.
• IndexDefs - not used in MyDAC, but fills on first call
3.4 Connecting to MySQL Server

This tutorial describes how to connect to MySQL using the TMyConnection component.

1. Requirements
2. General Information
3. Creating a Connection
   o 3.1 Connecting at Design-Time
     ▪ 3.1.1 Using TMyConnection 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 MyDAC 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 3306), the database name, and the username and password. To connect at runtime, add the MyAccess unit to the uses clause for Delphi or include the MyAccess.hpp header file for C++ Builder.

General Information

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

You can connect to MySQL in the Direct mode or using the MySQL client library. The connection mode is defined by the Direct property, which defaults to True. To access MySQL using the client library, set the property to False.

MyDAC can also work with the embedded MySQL server. For information about connecting to the embedded server, see Connecting to MySQL Embedded.

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 TMyConnection object in the TMyConnection Editor or Object Inspector.
1. Find the TMyConnection component in the MyDAC category on the Tool Palette.
2. Double-click the component. A new object will appear on the form. If this is the first time
that you create a TMyConnection object in this project, the object will be named MyConnection1.

Using TMyConnection Editor
1. Double-click the MyConnection1 object.
2. Specify the DNS name or IP address of the MySQL server in the Server edit box.
3. If you use a port other than the default port 3306, specify it in the Port edit box.
4. Specify the database name in the Database edit box.
5. Specify the username (root by default) in the Username edit box.
6. Specify the password (empty value by default) in the Password edit box.

Using Object Inspector
1. Select the MyConnection1 object on the form.
2. Set the Database property to the database name.
3. Set the Password property to the password (root by default).
4. If you use a port other than the default port 3306, set the Port property to your port.
5. Set the Server property to the DNS name or IP address of the MySQL server.
6. Set the Username property to the username (empty value by default).

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

**Delphi**

```pascal
var
  MyConnection1: TMyConnection;
begin
  MyConnection1 := TMyConnection.Create(nil);
  try
    // adds connection parameters
    MyConnection1.Server := 'server';
    MyConnection1.Database := 'database';
    MyConnection1.Username := 'username';
    MyConnection1.Password := 'password';
    MyConnection1.Port := 3306;
    // disables a login prompt
    MyConnection1.LoginPrompt := False;
    // opens a connection
    MyConnection1.Open;
  finally
    MyConnection1.Free;
  end;
end;
```

**C++ Builder**

```cpp
TMyConnection* MyConnection1 = new TMyConnection(NULL);
try {
```
// adds connection parameters
MyConnection1->Server = "server";
    MyConnection1->Database = "database";
    MyConnection1->Username = "username";
    MyConnection1->Password = "password";
    MyConnection1->Port = 3306;
    // disables a login prompt
    MyConnection1->LoginPrompt = False;
    // opens a connection
    MyConnection1->Open();
}__finally {
    MyConnection1->Free();
}

Opening a Connection

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

**Delphi**

MyConnection1.Open;

**C++Builder**

MyConnection1->Open;

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

**Delphi**

MyConnection1.Connected := True;

**C++ Builder**

MyConnection1->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 TMyConnection 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:
3.5 Connecting to MySQL Embedded

This tutorial describes how to connect to MySQL Embedded.
Contents
1. Requirements
2. General information
   - 2.1 Components for connection to MySQL Embedded Server
   - 2.2 Installation of MySQL Embedded Server
3. Creating Connection
   - 3.1 Design time creation
   - 3.2 Connection setting
   - 3.3 Run time creation
4. Opening connection
5. Closing connection
6. Modifying connection
7. Additional information
8. See Also

Requirements
In order to connect to MySQL Embedded Server, you need MySQL Embedded Server files, MyDAC installed, and IDE running. Also, if authentication is enabled in the settings of MySQL Embedded Server, you need to know the login and password (by default, authentication is disabled).

General information

Components for connection to MySQL Embedded Server
It is possible to connect to MySQL Embedded Server using both TMyEmbConnection and TMyConnection components. But, in contrast to TMyConnection, TMyEmbConnection doesn't require my.ini and allows to set connection with MySQL Embedded Server via the TMyEmbConnection.Params, TMyEmbConnection.BaseDir and TMyEmbConnection.DataDir properties.

Installation of MySQL Embedded Server
- Place libmysqld.dll to a directory with the application or in one of directories listed in the PATH environmental variable.
- In the directory with the application, create the share\english\ subdirectory and place there the errmsg.sys file - a MySQL file containing error messages in english.
• In the directory with the application, create the **data** subdirectory - a directory for database folders.
• In the "data" directory, create the **test** subdirectory - this one will be a **test** database folder that we will work with further.
• If you use TMyConnection, you will need the my.ini file, that should be located in the **C:\** or **C:\Windows** directory and edited by adding or replacing the following lines:

```ini
[embedded]
    basedir=./
    datadir=./data/
```

• If you need stored procedure functionality, user access rights, etc., copy the **mysql** folder to the **data** folder.

### Creating Connection

#### Design time creation

The following assumes that you have IDE running, and you are currently focused on the form designer.

*TMyEmbConnection:*
1. Open the Component palette and find the TMyEmbConnection component in the MyDAC category.
2. Double-click on the component. Note that a new object appears on the form. If this is the first time you create TMyEmbConnection in this application, it is named MyEmbConnection1.

*TMyConnection:*
1. Open the Component palette and find the TMyConnection component in the MyDAC category.
2. Double-click on the component. Note that a new object appears on the form. If this is the first time you create TMyConnection in this application, it is named MyConnection1.

After you have done these steps, you should set up the newly created MyEmbConnection1 or MyConnection1 component. You can do this in two ways:

#### Using Connection Editor

*TMyEmbConnection:*
1. Double-click on the created connection object (MyEmbConnection1 or MyConnection1).
2. For the MyConnection1 object enable the Embedded check box.
3. In the Database edit box specify the database name (for example, test). If the specified database does not exist, you will get an error on connection.

4. If authentication is enabled, specify the user name in the Username edit box and the password in the Password edit box.

Using Object Inspector

TMyEmbConnection:
1. Click on the created connection object (MyEmbConnection1 or MyConnection1) and press F11 to focus on the object properties.
2. For MyConnection1 object set Options.Embedded to True.
3. In the Database edit box specify the database name (for example, test). If the specified database does not exist, you will get an error on connection.
4. If authentication is enabled, specify the user name in the Username property and the password in the Password property.

Run time creation

The same operations performed in runtime look as follows:

TMyEmbConnection:

[Delphi]

procedure TMainForm.ButtonConnectClick(Sender: TObject);
var
    con: TMyEmbConnection;
begin
    con := TMyEmbConnection.Create(nil);
    try
        con.Database := 'test'; // if the specified database does not exist, you will get an error on connection.
        con.Password := 'password'; // if authentication is enabled
        con.Username := 'username'; // if authentication is enabled
        con.LoginPrompt := False; // to prevent showing of the connection dialog
        con.Open;
    finally
        con.Free;
    end;
end;

Note: To run this code, you have to add the MSCompactConnection and OLEDBAccess units to the USES clause of your unit.

[C++Builder]

void __fastcall TMainForm::ButtonConnectClick(TObject *Sender) {

© 2021 Devart
TMyEmbConnection* con = new TMyEmbConnection(NULL);
try
{
    con->Database = "test"; // if the specified database does not exist, you
    con->Password = "password"; // if authentication is enabled
    con->Username = "username"; // if authentication is enabled
    con->LoginPrompt = False; // to prevent showing of the connection dialog
    con->Open();
} finally
{
    con->Free();
}

Note: To run this code, you have to include the MyEmbConnection.hpp header file to your unit.

TMyConnection:

[Delphi]

procedure TMainForm.ButtonConnectClick(Sender: TObject);
var
    con: TMyConnection;
begin
    con := TMyConnection.Create(nil);
    try
        con.Options.Embedded := True; // enable embedded mode
        con.Database := 'test'; // if the specified database does not exist, you
        con.Password := 'password'; // if authentication is enabled
        con.Username := 'username'; // if authentication is enabled
        con.LoginPrompt := False; // to prevent showing of the connection dialog
        con.Open;
    finally
        con.Free;
    end;
end;

Note: To run this code, you have to add the MSAccess unit to the USES clause of your unit.

[C++ Builder]

void __fastcall TMainForm::ButtonConnectClick(TObject *Sender)
{
    TMyConnection* con = new TMyConnection(NULL);
    try
    {
        con->Options->Embedded = True; // enable embedded mode
        con->Database = "database"; // if the specified database does not exist,
        con->Password = "password"; // if authentication is enabled
        con->Username = "username"; // if authentication is enabled
    } finally
    {
        con->Free;
    }
}
con->LoginPrompt = False; // to prevent showing of the connection dialog
con->Open();

finally
{
    con->Free();
}

**Note:** To run this code, you have to include the MSAccess.hpp header file to your unit.

### Opening connection

As you can see above, opening connection at run-time is as simple as calling of the Open method:

**[Delphi]**

con.open;

**[C++ Builder]**

con->Open();

Another way to open connection at run-time is to set the Connected property to True:

**[Delphi]**

con.Connected := True;

**[C++ Builder]**

con->Connected = True;

This way can be used at design-time as well. Of course, the connection (TMyEmbConnection or TMyConnection) must have valid connection options assigned earlier. When you call Open, MyDAC tries to open the database. If any problem occurs, it raises an exception with brief explanation on what is wrong. If no problem is encountered and the database is opened, the Open method returns and the Connected property is changed to True.

### Closing connection

To close connection, call its Close method, or set its Connected property to False:

**[Delphi]**

con.close;

**[C++ Builder]**

con.Close();
or:

**[Delphi]**

```delphi
con.Connected := False;
```

**[C++ Builder]**

```cpp
con.Connected = False;
```

### Modifying connection

You can modify connection by changing the properties of the TMyEmbConnection or TMyConnection component. Keep in mind that while some of the properties can be altered freely, most of them close connection when a new value is assigned. For example, if you change the Username property, it is closed immediately, and you have to reopen it manually.

### Additional information

To connect to MySQL Embedded Server versions 5.6.x, it is necessary to set the MYSQLEmbDisableEventLog global variable (declared in the MySqlApi module) to True before opening the connection. You can read additional information about connecting to MySQL Embedded Server in the [Embedded Server](#) article.

MyDAC has a wide set of features you can take advantage of. The following list enumerates some of them, so you can explore the advanced techniques to achieve better performance, balance network load or enable additional capabilities:

- Connection Pooling
- Disconnected Mode
- Data Type Mapping

### See Also

- [TMyEmbConnection](#)
- [TMyConnection](#)
- [Server](#)
- [Database](#)
- [Username](#)
- [Password](#)
- [LoginPrompt](#)
3.6 Creating Database Objects

This tutorial describes how to create database objects in MySQL using the TMyCommand and TMyScript 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 MySQL and Connecting to MySQL Embedded). To create database objects at runtime, add the MyAccess and MyScript units to the uses clause for Delphi or include the MyAccess.hpp and MyScript.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 TMyCommand, or use GUI tools for databases like dbForge Studio for MySQL. This tutorial uses the data access components to create tables and stored procedures.

Creating Tables

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

Design-Time

- Find the TMyCommand component in the MyDAC category on the Tool Palette.
- Double-click the component. A new object will appear on the form. If this is the first TMyCommand object in this project, it will be named MyCommand1. Note that the Connection...
property is automatically set to an existing connection.

- Double-click the MyCommand1 object.
- Enter the following statements:

```delphi
CREATE TABLE dept (  
    deptno INT NOT null PRIMARY KEY,  
    dname VARCHAR(14),  
    loc VARCHAR(13)  
);  
CREATE TABLE emp (  
    empno INT NOT null PRIMARY KEY,  
    ename VARCHAR(10),  
    job VARCHAR(9),  
    mgr INT,  
    hiredate TIMESTAMP,  
    sal INT,  
    comm INT,  
    deptno INT REFERENCES dept (deptno)  
);  
```

- Click the Execute button to create two tables.

### Runtime

The same tables created at runtime:

**Delphi**

```delphi
var  
MyCommand1: TMyCommand;begin  
MyCommand := TMyCommand.Create(nil);  
try  
    // MyConnection1 (or MyEmbConnection1) was set up earlier  
    MyCommand1.Connection := MyConnection1;  
    //adds statements to create tables  
    MyCommand1.SQL.Add('CREATE TABLE dept (');  
    MyCommand1.SQL.Add('    deptno INT NOT NULL PRIMARY KEY,');  
    MyCommand1.Add('    dname VARCHAR(14),');  
    MyCommand1.SQL.Add('    loc VARCHAR(13)');  
    MyCommand1.SQL.Add(');  
    MyCommand1.SQL.Add('CREATE TABLE emp (');  
    MyCommand1.SQL.Add('    empno INT NOT NULL PRIMARY KEY,');  
    MyCommand1.SQL.Add('    ename VARCHAR(10),');  
    MyCommand1.SQL.Add('    job VARCHAR(9),');  
    MyCommand1.SQL.Add('    mgr INT,');  
    MyCommand1.SQL.Add('    hiredate TIMESTAMP,');  
    MyCommand1.SQL.Add('    sal INT,');  
    MyCommand1.SQL.Add('    comm INT,');  
    MyCommand1.SQL.Add('    deptno INT REFERENCES dept (deptno)');  
    MyCommand1.SQL.Add(');');```

© 2021 Devart
// executes the statements
MyCommand1.Execute;
finally
    MyCommand1.Free;
end;
end;

C++Builder

TMyCommand* MyCommand1 = new TMyCommand(NULL);
try {
    // MyConnection1 (or MyEmbConnection1) was set up earlier
    MyCommand1->Connection = MyConnection1;
    // adds statements to create tables
    MyCommand1->SQL->Add("CREATE TABLE dept (");
    MyCommand1->SQL->Add(" deptno INT NOT NULL PRIMARY KEY , ");
    MyCommand1->SQL->Add(" dname VARCHAR(14) , ");
    MyCommand1->SQL->Add(" loc VARCHAR(13) ");
    MyCommand1->SQL->Add(");
    MyCommand1->SQL->Add(" CREATE TABLE emp ( ");
    MyCommand1->SQL->Add(" empno INT NOT NULL PRIMARY KEY , ");
    MyCommand1->SQL->Add(" ename VARCHAR(10) , ");
    MyCommand1->SQL->Add(" job VARCHAR(9) , ");
    MyCommand1->SQL->Add(" mgr INT , ");
    MyCommand1->SQL->Add(" hiredate TIMESTAMP , ");
    MyCommand1->SQL->Add(" sal INT , ");
    MyCommand1->SQL->Add(" comm INT , ");
    MyCommand1->SQL->Add(" deptno INT REFERENCES dept (deptno) ");
    MyCommand1->SQL->Add(");
    // executes the statements
    MyCommand1->Execute();
}__finally {
    MyCommand1->Free();
}

Creating Stored Procedures

To create stored procedures, the TMyScript component is used in this tutorial. MySQL supports stored procedures since version 5.6.

Design-Time

- Find the TMyScript component in the MyDAC category on the Tool Palette.
- Double-click the component. A new object will appear on the form. If this is the first TMyScript object in this project, it will be named MyScript1. Note that the Connection property is already set to an existing connection.
- Double-click the MyScript1 object.
- Enter the following statement:

```
DELIMITER $$
CREATE PROCEDURE TenMostHighPaidEmployees()
```
BEGIN
  SELECT emp.ename, emp.sal FROM emp ORDER BY emp.sal DESC LIMIT 10;
END $$

CREATE PROCEDURE GetEmpNumberInDept (IN deptno INT, OUT empnumb INT)
BEGIN
  SELECT COUNT(*) INTO empnumb FROM EMP WHERE EMP.deptno = deptno;
END $$
DELIMITER ;

- Click the Execute button to create two stored procedures.

Runtime

The same stored procedures created at runtime:

**Delphi**

```delphi
var
  MyScript1: TMyScript;
begin
  MyScript1 := TMyScript.Create(nil);
  try
    // MyConnection1 (or MyEmbConnection1) was set up earlier
    IBCScript1.Connection := IBCConnection1;
    // adds statements to create procedures
    MyScript1.SQL.Add('DELIMITER $'$);
    MyScript1.SQL.Add('BEGIN');
    MyScript1.SQL.Add('SELECT emp.ename, emp.sal FROM emp ORDER BY emp.sal DESC LIMIT 10;
END $$';
    MyScript1.SQL.Add('CREATE PROCEDURE GetEmpNumberInDept (IN deptno INT, OUT empnumb INT)');
  MyScript1.SQL.Add('BEGIN');
    MyScript1.SQL.Add('SELECT COUNT(*) INTO empnumb FROM EMP WHERE EMP.deptno = deptno;');
    MyScript1.SQL.Add('END $$');
  MyScript1.SQL.Add('DELIMITER ;');
  // executes the statements
  MyScript1.Execute;
finally
  MyScript1.Free;
end;
end;
```

**C++Builder**

```cpp
TMyScript* MyScript1 = new TIBCScript(NULL);
try {
    // MyConnection1 (or MyEmbConnection1) was set up earlier
    MyScript1->Connection = IBCConnection1;
    // adds statements to create procedures
    MyScript1->SQL->Add("DELIMITER $$");
    MyScript1->SQL->Add("BEGIN");
    MyScript1->SQL->Add("SELECT emp.ename, emp.sal FROM emp ORDER BY emp.sal DESC LIMIT 10;
    MyScript1->SQL->Add("END $$");
    MyScript1->SQL->Add("CREATE PROCEDURE GetEmpNumberInDept (IN deptno INT, OUT empnumb INT)";
    MyScript1->SQL->Add("BEGIN");
    MyScript1->SQL->Add("SELECT COUNT(*) INTO empnumb FROM EMP WHERE EMP.deptno = deptno;
    MyScript1->SQL->Add("END $$");
    // executes the statements
    MyScript1->Execute;
    finally
    MyScript1->Free;
}
```
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, TMyCommand can be used to insert statements one by one, while TMyScript is intended to execute multiple DDL/DML statements as a single SQL script. For more information on DDL statements, refer to the MySQL documentation.

3.7 Deleting Data From Tables

This tutorial describes how to delete data from tables using the TMyQuery and TMyTable components.

1. Requirements
2. General information
3. Using DataSet Functionality
4. Building DML Statements Manually
   - 4.1 DML Statements With Parameters
   - 4.2 DML Statements As Plain Text
5. Additional Information

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded"), how to create necessary objects on the server (tutorial "Creating Database Objects"), and how to insert data to created tables (tutorial "Inserting Data Into Tables").

General information

Data on server can be deleted using Data Manipulation Language (DML), which is a part of
SQL. DML statements can be executed on server by an account that has necessary privileges. There are two ways to manipulate a database. You can build DML statements manually and run them within some component like TMyQuery. Another way is to use the dataset functionality (the Delete method) of the TMyQuery and TMyTable components. We will discuss both ways. The goal of this tutorial is to delete a record in the table `dept`.

### Using DataSet Functionality

The Delete method of the TMyQuery and TMyTable components allows deleting data without using DML statements. DML statements are generated by MyDAC components internally. The code below demonstrates using this method:

**[Delphi]**

```delphi
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // retrieve data
    q.SQL.Text := 'SELECT * FROM dept';
    q.Open;
    // delete the current record
    q.Delete;
  finally
    q.Free;
  end;
end;
```

**[C++Builder]**

```cpp
{  TMyQuery* q = new TMyQuery(NULL);
  try
  {    // con is either TMyConnection or TMyEmbConnection already set up
    q->Connection = con;
    // retrieve data
    q->SQL->Text = "SELECT * FROM dept";
    q->Open();
    // delete the current record
    q->Delete();
  }
  finally
  {    q->Free();
  }
}
Building DML Statements Manually

DML Statements can contain plain text and text with parameters. This section describes both ways.

DML Statements With Parameters

[Delphi]

```delphi
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // set SQL query for delete record
    q.SQL.Clear;
    q.SQL.Add('DELETE FROM dept WHERE deptno = :deptno;');
    // set parameters
    q.ParamByName('deptno').AsInteger := 10;
    // execute query
    q.Execute;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```c++
{
  TMyQuery* q = new TMyQuery(NULL);
  try
  {
    // con is either TMyConnection or TMyEmbConnection already set up
    q->Connection = con;
    // set SQL query for delete record
    q->SQL->Clear();
    q->SQL->Add("DELETE FROM dept WHERE deptno = :deptno;"室;
    // set parameters
    q->ParamByName("deptno")->AsInteger = 10;
    // execute query
    q->Execute();
  }
  __finally
  {
    q->Free();
  }
}
```

DML Statements As Plain Text
[Delphi]

var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // set SQL query for delete record
    q.SQL.Clear;
    q.SQL.Add('DELETE FROM dept WHERE deptno = 10;');
    // execute query
    q.Execute;
  finally
    q.Free;
  end;
end;

[C++Builder]

{
    TMyQuery* q = new TMyQuery(NULL);
    try
    {
        // con is either TMyConnection or TMyEmbConnection already set up
        q->Connection = con;
        // set SQL query for delete record
        q->SQL->Clear();
        q->SQL->Add("DELETE FROM dept WHERE deptno = 10;" );
        // execute query
        q->Execute();
    }
    finally
    {
        q->Free();
    }
}

Additional Information

It is also possible to use stored procedures for deleting data. In this case, all data 
manipulation logic is defined on server. You can find more about using stored procedures in 
the tutorial "Stored Procedures".

© 1997-2021
Devart. All Rights Reserved.
3.8 Inserting Data Into Tables

This tutorial describes how to insert data into tables using the TMyQuery and TMyTable components.

1. Requirements
2. General information
3. Design Time
4. Run Time
   - 4.1 Using DataSet Functionality
   - 4.2 Building DML Statements Manually
     - 4.2.1 DML Statements With Parameters
     - 4.2.2 DML Statements As Plain Text
5. Additional Information

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded") and that necessary objects are already created on the server (tutorial "Creating Database Objects").

General information

Data on server can be inserted using Data Manipulation Language (DML), which is a part of SQL. DML statements can be executed on server by an account that has necessary privileges. There are two ways to manipulate a database. You can build DML statements manually and run them within some component like TMyQuery. Another way is to use the dataset functionality (the Insert, Append, and Post methods) of the TMyQuery and TMyTable components. We will discuss both ways.

The goal of this tutorial is to insert the following data into tables dept and emp:

**Table dept**

<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>CHIC</td>
</tr>
</tbody>
</table>
### Table emp

<table>
<thead>
<tr>
<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>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>ALLE</td>
<td>SALESMAN</td>
<td>7698</td>
<td>20.02.1981</td>
<td>1600</td>
<td>300</td>
<td>30</td>
</tr>
<tr>
<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>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>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>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>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>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>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>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>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>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>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>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>

**Note:** The empno field of the emp table is an IDENTITY(1,1) (i.e. autoincrement) field, so its value is filled automatically by the server.
Design time

- Open the Component palette and find the TMyQuery component in the MyDAC category.
- Double-click on the component. Note that a new object appears on the form. If this is the first time you create TMyQuery in this application, it is named MSQuery1. Note that the MSQuery1.Connection property is already set to an existent (on the form) connection.
- Double-click on the MSQuery1 object.
- Type the following lines:

```
INSERT INTO dept VALUES (10,'ACCOUNTING','NEW YORK');
```
- Press the Execute button.

Performing these steps adds a new record to the dept table.

Run time

Using DataSet Functionality

The Insert, Append, and Post methods of the TMyQuery and TMyTable components allow inserting data not using DML statements. DML statements are generated by MyDAC components internally. The difference between the Append and Insert methods is that Append creates a new empty record in the end of a dataset, when Insert creates it in the position of the current record of a dataset. The code below demonstrates using these methods:

[Delphi]

```delphi
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    q.Connection := con; // con is either TMyConnection or TMyEmbConnection
    q.SQL.Text := 'SELECT * FROM dept';
    q.Open;
    q.Append;
    q.FieldByName('deptno').AsInteger := 10;
    q.FieldByName('dname').AsString := 'ACCOUNTING';
    q.FieldByName('loc').AsString := 'NEW YORK';
    q.Post;
  finally
    q.Free;
  end;
end;
```

[C++Builder]

```cpp
{
```
Building DML Statements Manually

DML Statements can contain plain text and text with parameters. This section describes both ways.

DML Statements With Parameters

[Delphi]

```delphi
var  q: TMyQuery;
begin  q := TMyQuery.Create(nil);
  try    q.Connection := con; // con is either TMyConnection or TMyEmbConnection
    q.SQL.Text := "SELECT * FROM dept";
    q.Open;
    q.Append;
    q.FieldByName("deptno")\->AsInteger = 10;
    q.FieldByName("dname")\->AsString = "ACCOUNTING";
    q.FieldByName("loc")\->AsString = "NEW YORK";
    q.Post;
  finally    q.Free;
  end;
end;
```

[C++Builder]

```c++
{  TMyQuery* q = new TMyQuery(NULL);
  try  {    q->Connection = con; // con is either TMyConnection or TMyEmbConnection
    q->SQL->Clear();
    q->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (:deptno, :dname, :loc)
      ->AsInteger = 10;
    q->ParamByName("dname")\->AsString = 'ACCOUNTING';
    q->ParamByName("loc")\->AsString = 'NEW YORK';
    q->Execute;
  finally    q->Free;
  }
}
```
DML Statements As Plain Text

**[Delphi]**

```delphi
var  q: TMyQuery;
begin  q := TMyQuery.Create(nil);
  try    q.Connection := con; // con is either TMyConnection or TMyEmbConnection
    q.SQL.Clear;
    q.SQL.Add('INSERT INTO dept(deptno, dname, loc) VALUES (10,''ACCOUNTING'',''NEW YORK'');');    q.Execute;
  finally    q.Free;
  end;
end;
```

**[C++Builder]**

```cpp
{  TMyQuery* q = new TMyQuery(NULL);
  try  {
    q->Connection = con; // con is either TMyConnection or TMyEmbConnection
    q->SQL->Clear();
    q->SQL->Add("INSERT INTO dept(deptno, dname, loc) VALUES (10,'ACCOUNTING','NEW YORK');
    q->Execute();
  }  finally  {
    q->Free();
  }
}
```

Additional Information

Actually, there are lots of ways to insert data into tables. Any tool or component capable of running a SQL query can be used to manage data. Some components are better for performing certain tasks. For example, **TMyLoader** is the fastest way to insert data, **TMyScript** is designed for executing series of statements one by one.

It is also possible to use stored procedures for inserting data. In this case, all data
3.9 Retrieving Data

This tutorial describes how to retrieve data from tables using the TMyQuery and TMyTable components.

1. Requirements
2. General Information
3. TMyQuery
4. TMyTable
5. Additional information

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded Server"), how to create necessary objects on the server (tutorial "Creating Database Objects"), and how to insert data to created tables (tutorial "Inserting Data Into Tables").

General information

As we know, an original function of any database application is establishing connection to a data source and working with data contained in it. MyDAC provides several components that can be used for data retrieving, such as TMyQuery and TMyTable. We will discuss data retrieving using these components.

The goal of this tutorial is to retrieve data from a table dept.

TMyQuery

The following code demonstrates retrieving of data from the dept table using the TMyQuery component:

[Delp hi]

```delphi
var
  q: TMyQuery;
begin
```

manipulation logic is defined on the server. You can find more about using stored procedures in the tutorial "Stored Procedures".
q := TMyQuery.Create(nil);
try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // retrieve data
    q.SQL.Text := 'SELECT * FROM dept';
    q.Open;
    // shows the number of records obtained from the server
    ShowMessage(IntToStr(q.RecordCount));
finally
    q.Free;
end;

[C++Builder]

{
    TMyQuery* q = new TMyQuery(NULL);
    try
    {
        // con is either TMyConnection or TMyEmbConnection already set up
        q->Connection = con;
        // retrieve data
        q->SQL->Text = "SELECT * FROM dept";
        q->Open();
        // shows the number of records obtained from the server
        ShowMessage(IntToStr(q->RecordCount));
    }
    finally
    {
        q->Free();
    }
}

TMyTable

The following code demonstrates retrieving of data from the dept table using the TMyTable component:

[Delphi]

var
    tbl: TMyTable;
begin
    tbl := TMyTable.Create(nil);
    try
        // con is either TMyConnection or TMyEmbConnection already set up
        tbl.Connection := con;
        // retrieve data
        tbl.TableName := 'dept';
        tbl.Open;
        // shows the number of records obtained from the server
        ShowMessage(IntToStr(tbl.RecordCount));
    finally
        tbl.Free;
    end;
Additional Information

It is also possible to use stored procedures for data retrieving. In this case, all data manipulation logic is defined on server. You can find more about using stored procedures in the tutorial "Stored Procedures".

3.10 Modifying Data

This tutorial describes how to modify data in tables using the TMyQuery and TMyTable components.

1. Requirements
2. General information
3. Using DataSet Functionality
4. Building DML Statements Manually
   4.1 DML Statements With Parameters
   4.2 DML Statements As Plain Text
5. Additional Information

Requirements
This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded"), how to create necessary objects on the server (tutorial "Creating Database Objects"), and how to insert data to created tables (tutorial "Inserting Data Into Tables").

General information

Data on server can be modified using Data Manipulation Language (DML), which is a part of SQL. DML statements can be executed on server by an account that has necessary privileges. There are two ways to manipulate a database. You can build DML statements manually and run them within some component like TMyQuery. Another way is to use the dataset functionality (the Edit and Post methods) of the TMyQuery and TMyTable components. We will discuss both ways. The goal of this tutorial is to modify the following record of the table `dept`:

<table>
<thead>
<tr>
<th></th>
<th>ACCOUNTING</th>
<th>NEW YORK</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

to make it look as follows:

<table>
<thead>
<tr>
<th></th>
<th>RESEARCH</th>
<th>LOS ANGELES</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using DataSet Functionality

The Edit and Post methods of the TMyQuery and TMyTable components allow deleting data without using DML statements. DML statements are generated by MyDAC components internally. The code below demonstrates using these methods:

```delphi
var q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    try
      // retrieve data
      q.SQL.Text := 'SELECT * FROM dept';
      q.Open;
      // to make the record with deptno=10 the current record
      q.FindKey([10]);
      // modify record
      q.Edit;
      q.FieldByName('dname').AsString := 'RESEARCH';
      q.FieldByName('loc').AsString := 'LOS ANGELES';
      q.Post;
  finally
  end;
```
Building DML Statements Manually

DML Statements can contain plain text and text with parameters. This section describes both ways.

DML Statements With Parameters

**[Delphi]**

```delphi
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // set SQL query for update record
    q.SQL.Clear;
    q.SQL.Add('UPDATE dept SET dname = :dname, loc = :loc WHERE deptno = :deptno;');
    q.ParamByName('deptno').AsInteger := 10;
    q.ParamByName('dname').AsString := 'RESEARCH';
    q.ParamByName('loc').AsString := 'LOS ANGELES';
    // execute query
    q.Execute;
  finally
    q.Free;
  end;
end;
```

**[C++Builder]**

```cpp
{  TMyQuery* q = new TMyQuery(NULL);
  try  {
    // con is either TMyConnection or TMyEmbConnection already set up
    q->Connection = con;
    // retrieve data
    q->SQL->Text = "SELECT * FROM dept";
    q->Open();
    // to make the record with deptno=10 the current record
    q->FindKey(ARRAYOFCONST((10)));
    // modify record
    q->Edit();
    q->FieldByName("dname")->AsString = "RESEARCH";
    q->FieldByName("loc")->AsString = "LOS ANGELES";
    q->Post();
  }  __finally  {
    q->Free();
  }
}  
```

Building DML Statements Manually

DML Statements can contain plain text and text with parameters. This section describes both ways.

DML Statements With Parameters

**[Delphi]**

```delphi
var
  q: TMyQuery;
begin
  q := TMyQuery.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    q.Connection := con;
    // set SQL query for update record
    q.SQL.Clear;
    q.SQL.Add('UPDATE dept SET dname = :dname, loc = :loc WHERE deptno = :deptno;');
    q.ParamByName('deptno').AsInteger := 10;
    q.ParamByName('dname').AsString := 'RESEARCH';
    q.ParamByName('loc').AsString := 'LOS ANGELES';
    // execute query
    q.Execute;
  finally
    q.Free;
  end;
end;
```

**[C++Builder]**

```cpp
{  TMyQuery* q = new TMyQuery(NULL);
  try  {
    // con is either TMyConnection or TMyEmbConnection already set up
    q->Connection = con;
    // retrieve data
    q->SQL->Text = "SELECT * FROM dept";
    q->Open();
    // to make the record with deptno=10 the current record
    q->FindKey(ARRAYOFCONST((10)));
    // modify record
    q->Edit();
    q->FieldByName("dname")->AsString = "RESEARCH";
    q->FieldByName("loc")->AsString = "LOS ANGELES";
    q->Post();
  }  __finally  {
    q->Free();
  }
}  
```
finally
    q.Free;
end;
end;

[C++Builder]
{
    TMyQuery* q = new TMyQuery(NULL);
    try
    {
        // con is either TMyConnection or TMyEmbConnection already set up
        q->Connection = con;
        // set SQL query for update record
        q->SQL->Clear();
        q->SQL->Add("UPDATE dept SET dname = :dname, loc = :loc WHERE deptno = :deptno;" "RESEARCH", "LOS ANGELES");
        // execute query
        q->Execute();
    }
    __finally
    {
        q->Free();
    }
}

DML Statements As Plain Text

[Delphi]
var
    q: TMyQuery;
begin
    q := TMyQuery.Create(nil);
    try
        // con is either TMyConnection or TMyEmbConnection already set up
        q.Connection := con;
        // set SQL query for update record
        q.SQL.Clear;
        q.SQL.Add('UPDATE dept SET dname = ''RESEARCH'', loc = ''LOS ANGELES'' WHERE deptno = 10;');
        // execute query
        q.Execute;
    finally
        q.Free;
    end;
end;

[C++Builder]
{
    TMyQuery* q = new TMyQuery(NULL);
    try
    {
        // con is either TMyConnection or TMyEmbConnection already set up
        q->Connection = con;
        // set SQL query for update record
        q->SQL->Clear();
        q->SQL->Add("UPDATE dept SET dname = :dname, loc = :loc WHERE deptno = :deptno;" "RESEARCH", "LOS ANGELES");
        // execute query
        q->Execute();
    }
    __finally
    {
        q->Free();
    }
}
Additional Information

It is also possible to use stored procedures for modifying data. In this case, all data manipulation logic is defined on server. You can find more about using stored procedures in the tutorial "Stored Procedures".

3.11 Stored Procedures

This tutorial describes how to work with stored procedures using the TMyStoredProc component and insert data into tables.

1. Requirements
2. General information
3. Input parameters
4. Output parameters
5. Input/output parameters
6. Using MySQL Stored Functions
7. Returning result sets

Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded"), how to create necessary objects on the server (tutorial "Creating Database Objects"), and how to insert data to created tables (tutorial "Inserting Data Into Tables").

General information
A **stored procedure** is a database object that consists of a set of SQL statements, grouped together, stored in the database, and run as a unit to solve a specific problem or perform a set of related tasks. Procedures let you combine the ease and flexibility of SQL with the procedural functionality of a structured programming language. Large or complex processing that might require execution of several SQL statements is moved into stored procedures, and all applications call the procedures only.

Objects similar to stored procedures are **stored functions**. Almost everything that is true for procedures, holds for functions as well. The main difference between these objects is that function has a return value, and procedure has not. Also, stored procedures may have input, output, and input/output parameters.

**Note:** stored procedures and stored functions are supported since MySQL 5.0.

## Input parameters

**Input parameter (IN)** is a parameter which value is passed into a stored procedure/function module. The procedure might modify the value, but the modification is not visible to the caller when the procedure is returned. The following procedure inserts a new row into the table `dept`:

```sql
CREATE PROCEDURE InsertDept(
    IN p_deptno INT,
    IN p_dname VARCHAR(14),
    IN p_loc VARCHAR(13)
) BEGIN
    INSERT INTO dept(deptno, dname, loc) VALUES(p_deptno, p_dname, p_loc);
END;
```

It needs to receive the values to be inserted into the new record, and thus the procedure has three input parameters, corresponding to each field of the table. This procedure may be executed as follows:

```sql
call InsertDept(10, 'ACCOUNTING', 'NEW YORK');
```

To execute the `InsertDept` stored procedure using the `TMyStoredProc` component, the following code can be used:

**[Delphi]**

```delphi
var
    sp: TMyStoredProc;
begin
    sp := TMyStoredProc.Create(nil);
    try
        // con is either TMyConnection or TMyEmbConnection already set up
```
sp.Connection := con;
// choose a stored procedure name to execute
sp.StoredProcName := 'InsertDept';
// build a query for a chosen stored procedure based on the Params and S
sp.PrepareSQL;
// assign parameter values
sp.ParamByName('p_deptno').AsInteger := 10;
sp.ParamByName('p_dname').AsString := 'ACCOUNTING';
sp.ParamByName('p_loc').AsString := 'NEW YORK';
// execute the stored procedure
sp.Execute;
finally
  sp.Free;
end;
end;

[C++Builder]

{  TMyStoredProc* sp = new TMyStoredProc(NULL);  try  {    // con is either TMyConnection or TMyEmbConnection already set up    sp->Connection = con;
    // choose a stored procedure name to execute    sp->StoredProcedureName = "InsertDept";
    // build a query for chosen stored procedure based on the Params and Stor
    sp->PrepareSQL();
    // assign parameter values
    sp->ParamByName("p_deptno")-&gt;AsInteger = 10;
    sp->ParamByName("p_dname")-&gt;AsString = "ACCOUNTING";
    sp->ParamByName("p_loc")-&gt;AsString = "NEW YORK";
    // execute the stored procedure
    sp-&gt;Execute();
  }  __finally  {    sp-&gt;Free();  }
}

Output parameters

Output parameter (OUT) is a parameter which value is passed out of the stored procedure/ function module. Its initial value is NULL within the procedure, and its value is visible to the caller when the procedure is returned. The following stored procedure returns the count of records in the table dept:

CREATE PROCEDURE CountDept (  OUT cnt INT  )  BEGIN  SELECT count(*) FROM dept into cnt;  END;
To execute the CountDept stored procedure using the TMyStoredProc component, the following code can be used:

**[Delphi]**

```delphi
var  sp: TMyStoredProc;
begin  sp := TMyStoredProc.Create(nil);
  try    // con is either TMyConnection or TMyEmbConnection already set up
          sp.Connection := con;
    // choose a stored procedure name to execute
    sp.StoredProcedureName := 'CountDept';
    // build a query for chosen stored procedure based on the Params and Store
    sp.PrepareSQL;
    // execute the stored procedure
    sp.Execute;
    // show the value of the output parameter
    ShowMessage(IntToStr(sp.ParamByName('cnt').AsInteger));
  finally
    sp.Free;
  end;
end;
```

**[C++Builder]**

```cpp
{  TMyStoredProc* sp = new TMyStoredProc(NULL);
  try  {
    // con is either TMyConnection or TMyEmbConnection already set up
    sp->Connection = con;
    // choose a stored procedure name to execute
    sp->StoredProcedureName = "CountDept";
    // build a query for chosen stored procedure based on the Params and Store
    sp->PrepareSQL();
    // execute the stored procedure
    sp->Execute();
    // show the value of the output parameter
    ShowMessage(IntToStr(sp->ParamByName("cnt")->AsInteger));
  }
  finally
  {
    sp->Free();
  }
}
```

**Input/output parameters**

An **input/output parameter (INOUT)** is a parameter that functions as an IN or an OUT parameter, or both. The value of the INOUT parameter is passed into the stored procedure/function and a new value can be assigned to the parameter and passed out of the module. An
IN/OUT parameter must be a variable, not a constant. It can be found on both sides of an assignment. In other words, an IN/OUT parameter behaves like an initialized variable.

For example, the following stored procedure returns the salary with five percents bonus:

```
CREATE PROCEDURE GiveBonus(INOUT sal FLOAT)
BEGIN
  SET sal = sal * 1.05;
END
```

To execute the GiveBonus stored procedure using the TMyStoredProc component, the following code can be used:

**[Delphi]**

```delphi
var
  sp: TMyStoredProc;
begin
  sp := TMyStoredProc.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    sp.Connection := con;
    // choose a stored procedure name to execute
    sp.StoredProcedureName := 'GiveBonus';
    // build a query for chosen stored procedure based on the Params and Storage
    sp.PreparedStatement;
    // assign parameter values
    sp.ParamByName('sal').AsFloat := 500.5;
    // execute the stored procedure
    sp.Execute;
    // show the value of the input/output parameter
    ShowMessage(sp.ParamByName('sal').AsString);
  finally
    sp.Free;
  end;
end;
```

**[C++Builder]**

```cpp
{  TMyStoredProc* sp = new TMyStoredProc(NULL);
  try  {
    // con is either TMyConnection or TMyEmbConnection already set up
    sp->Connection = con;
    // choose a stored procedure name to execute
    sp->StoredProcedureName = "GiveBonus";
    // build a query for chosen stored procedure based on the Params and Storage
    sp->PrepareSQL();
    // assign parameter values
    sp->ParamByName("sal")->AsFloat = 500.5;
    // execute the stored procedure
    sp->Execute();
    // show the value of the input/output parameter
    ShowMessage(sp->ParamByName("sal")->AsString);
  } finally  {
    sp->Free();
  }
}
```
Using MySQL Stored Functions

The tasks described above can be performed using stored functions. For example, the following stored function returns the bonus salary like the GiveBonus stored procedure:

```sql
CREATE FUNCTION GiveBonusFunc(sal FLOAT) RETURNS FLOAT
BEGIN
  RETURN sal * 1.05;
END
```

This function may be executed as follows:

```sql
select GiveBonusFunc(500.5);
```

To execute the GiveBonusFunc stored function using the TMy StoredProc component, the following code can be used:

**[Delphi]**

```delphi
var sp: TMyStoredProc;
begin
  sp := TMyStoredProc.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    sp.Connection := con;
    // choose a stored function name to execute
    sp.StoredProcedureName := 'GiveBonusFunc';
    // build a query for chosen stored procedure based on the Params and Store
    sp.PrepareSQL;
    // assign parameter values
    sp.ParamByName('sal').AsFloat := 500.5;
    // execute the stored procedure
    sp.Execute;
    // show the returned value
    ShowMessage(sp.ParamByName('result').AsString);
  finally
    sp.Free;
  end;
end;
```

**[C++Builder]**

```c++
{
  TMyStoredProc* sp = new TMyStoredProc(NULL);
  try
  {
```
Note: To retrieve the result returned by the stored function using TMyStoredProc, the 'result' parameter created automatically should be used.

### Returning result sets

Besides scalar variables, a stored procedure can return result sets, i.e. the results of a SELECT statement. This question is discussed in details in the tutorial "Working With Result Sets Using Stored Procedures".

#### 3.12 Working With Result Sets Using Stored Procedures

This tutorial describes how to retrieve and modify result sets obtained from stored procedures using the TMyStoredProc component.

### Requirements

This walkthrough supposes that you know how to connect to server (tutorials "Connecting To MySQL" and "Connecting To MySQL Embedded"), how to create necessary objects on the server (tutorial "Creating Database Objects"), and how to insert data to created tables (tutorial "Inserting Data Into Tables").

### General information

Besides scalar variables, stored procedures can return result sets, i.e. the results of SELECT
Data can be inserted or modified in obtained result sets using the dataset functionality of the TMyStoredProc component.

The goal of this tutorial is to retrieve and modify data from the `dept` table using the TMyStoredProc component. The following stored procedure will be used to retrieve data:

```sql
CREATE PROCEDURE SelectDept()
BEGIN
  SELECT * FROM dept;
END
```

**Using DataSet functionality**

The Insert, Append, Edit, and Post methods of the TMyStoredProc component can be used to insert and modify data in obtained result sets. DML statements are generated by TMyStoredProc internally. The code below demonstrates using these methods:

```delphi
var
  sp: TMyStoredProc;
begin
  sp := TMyStoredProc.Create(nil);
  try
    // con is either TMyConnection or TMyEmbConnection already set up
    sp.Connection := con;
    // choose a stored procedure name
    sp.StoredProcedureName := 'SelectDept';
    // build a query for a chosen stored procedure based on the Params and SPS
    sp.PrepareSQL;
    // retrieve data
    sp.Open;
    // append record
    sp.Append;
    sp.FieldByName('deptno').AsInteger := 50;
    sp.FieldByName('dname').AsString := 'SALES';
    sp.FieldByName('loc').AsString := 'NEW YORK';
    sp.Post;
    // insert record
    sp.Insert;
    sp.FieldByName('deptno').AsInteger := 60;
    sp.FieldByName('dname').AsString := 'ACCOUNTING';
    sp.FieldByName('loc').AsString := 'LOS ANGELES';
    sp.Post;
    // to make the record with deptno=10 the current record
    sp.Findkey([10]);
    // modify record
    sp.Edit;
    sp.FieldByName('dname').AsString := 'RESEARCH';
    sp.FieldByName('loc').AsString := 'LOS ANGELES';
    sp.Post;
  finally
```
[C++Builder]

```cpp
sp = new TMyStoredProc(NULL);
try
{
    con is either TMyConnection or TMyEmbConnection already set up
    choose a stored procedure name
    sp->StoredProcName = "SelectDept";
    build a query for a chosen stored procedure based on the Params and SQL
    sp->PreparesQL();
    retrieve data
    sp->Open();
    append record
    sp->Append();
    sp->FieldByName("deptno")->AsInteger = 50;
    sp->FieldByName("dname")->AsString = "SALES";
    sp->FieldByName("loc")->AsString = "NEW YORK";
    sp->Post();
    insert record
    sp->Insert();
    sp->FieldByName("deptno")->AsInteger = 60;
    sp->FieldByName("dname")->AsString = "ACCOUNTING";
    sp->FieldByName("loc")->AsString = "LOS ANGELES";
    sp->Post();
    to make the record with deptno=10 the current record
    sp->FindKey(ARRAYOFCONST((10)));
    modify record
    sp->Edit();
    sp->FieldByName("dname")->AsString = "RESEARCH";
    sp->FieldByName("loc")->AsString = "LOS ANGELES";
    sp->Post();
}
finally
{
    sp->Free();
}
```

3.13 Demo Projects

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

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

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

Where are the MyDAC demo projects located?

In most cases all the MyDAC demo projects are located in "%MyDac%\Demos".

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

|--MyDacDemo [The main MyDAC demo project]
|--TechnologySpecific
| |-- Embedded [Win32 version of the Embedded MySQL server using demo]
| |-- 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
| |-- [Some other demo projects on design technologies]

MyDacDemo is the main demo project that shows off all the MyDAC functionality. The other directories contain a number of supplementary demo projects that describe special use cases. A list of all the samples in the MyDAC demo project and a description for the supplementary projects is provided in the following section.

Note: This documentation describes ALL the MyDAC demo projects. The actual demo projects you will have installed on your computer depends on your MyDAC version, MyDAC
Getting Started

Instructions for using the MyDAC demo projects

To explore a MyDAC demo project,
1. Launch your IDE.
2. In your IDE, choose File | Open Project from the menu bar.
3. Find the directory you installed MyDAC 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 MySQL. 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 MyDAC demo directory includes two sample SQL scripts for creating and dropping all the test schema objects used in the MyDAC 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 MyDAC demo project in the Getting Started topic. The other MyDAC demo projects include a ReadMe.txt file with individual building and launching instructions.

Demo project descriptions

MyDacDemo

MyDacDemo 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 MyDAC components.

General demos

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

**MySQL-specific demos**

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

*MyDacDemo* can be opened from %MyDac%/Demos/MyDacDemo/MyDacDemo.dpr (.bdsproj). The following table describes all demos contained in this project.

## Working with Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command</strong></td>
<td>Uses <strong>TMyCommand</strong> to execute SQL statements. Demonstrates how to execute commands in a separate thread, and how to break long-duration query execution.</td>
</tr>
<tr>
<td><strong>ConnectDialog</strong></td>
<td>Demonstrates how to customize the <a href="https://www.devart.com/mydac/docs/connectdialog">MyDAC connect dialog</a>. Changes the standard MyDAC 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 MyDAC connect dialog class.</td>
</tr>
<tr>
<td><strong>CRDBGrid</strong></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><strong>Loader</strong></td>
<td>Uses the <strong>TMyLoader</strong> component to quickly load data into a server table. TMyLoader loads data by grouping several data rows into a single INSERT statement and executing this statement. This way is much faster then executing one INSERT statement per row. This demo also compares the two TMyLoader data loading handlers: <code>GetColumnData</code> and <code>PutData</code>.</td>
</tr>
<tr>
<td><strong>Query</strong></td>
<td>Demonstrates working with <strong>TMyQuery</strong>, which is one of the most useful MyDAC components. Includes many TMyQuery usage scenarios. Demonstrates how to edit data and export it to XML files. Note: This is a very good introductory demo. We recommend starting here when first becoming familiar with MyDAC.</td>
</tr>
<tr>
<td><strong>StoredProc</strong></td>
<td>Uses <strong>TMyStoredProc</strong> to access an editable recordset from an MySQL stored procedure in the client application.</td>
</tr>
<tr>
<td><strong>Table</strong></td>
<td>Demonstrates how to use <strong>TMyTable</strong> 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><strong>UpdateSQL</strong></td>
<td>Demonstrates using the <strong>TMyUpdateSQL</strong> component to customize update commands. Lets you optionally use <strong>TMyCommand</strong> and <strong>TMyQuery</strong> objects for carrying out insert, delete, query, and update commands.</td>
</tr>
</tbody>
</table>
### VirtualTable

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.

### General Demos

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CachedUpdates</strong></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><strong>FilterAndIndex</strong></td>
<td>Demonstrates MyDAC'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><strong>MasterDetail</strong></td>
<td>Uses MyDAC 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><strong>Pictures</strong></td>
<td>Uses MyDAC functionality to work with BLOB fields and graphics. The sample demonstrates how to retrieve binary data from MySQL database and display it on visual components. The sample also shows how to load and save pictures to files and to the database.</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>Uses MyDAC functionality to work with text. The sample demonstrates how to retrieve text data from MySQL database and display it on visual components. The sample also shows how to load and save text to files and to the database.</td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td>Demonstrates the recommended approach for managing transactions with the TMyConnection component. The TMyConnection interface provides a wrapper for MySQL server commands like START TRANSACTION, COMMIT, ROLLBACK.</td>
</tr>
</tbody>
</table>

### MySQL-specific Demos

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lock</strong></td>
<td>Demonstrates two kinds of row-level locking (immediate locking and delayed locking) with the InnoDB storage engine. This functionality is based on the following MySQL commands: SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE.</td>
</tr>
</tbody>
</table>

### Supplementary Demo Projects
MyDAC also includes a number of additional demo projects that describe some special use cases, show how to use MyDAC in different IDEs and give examples of how to integrate it with third-party components. These supplementary MyDAC demo projects are sorted into subfolders in the %MyDac%\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 MyDAC can be used with FastReport components. This project consists of two parts. The first part is several packages that integrate MyDAC components into the FastReport editor. The second part is a demo application that lets you design and preview reports with MyDAC technology in the FastReport editor.</td>
</tr>
<tr>
<td></td>
<td>InfoPower</td>
<td>Uses InfoPower components to display recordsets retrieved with MyDAC. This demo project displays an InfoPower grid component and fills it with the result of an MyDAC query. Shows how to link MyDAC data sources to InfoPower components.</td>
</tr>
<tr>
<td></td>
<td>IntraWeb</td>
<td>A collection of sample projects that show how to use MyDAC components as data sources for IntraWeb applications. Contains IntraWeb samples for setting up a connection, querying a database and modifying data and working with CachedUpdates and MasterDetail relationships.</td>
</tr>
<tr>
<td></td>
<td>QuickReport</td>
<td>Lets you launch and view a QuickReport application based on MyDAC. This demo project lets you modify the application in design-time.</td>
</tr>
<tr>
<td></td>
<td>ReportBuilder</td>
<td>Uses MyDAC data sources to create a ReportBuilder report that takes data from MySQL database. Shows how to set up a ReportBuilder document in design-time and how to integrate MyDAC components into the Report Builder editor to perform document design in run-time.</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>Demonstrates working with Embedded MySQL server by using the TMyEmbConnection component. This demo creates a database structure, if it does not already exist, opens a table from this database. Also this demo shows how to process the log messages of the Embedded server.</td>
</tr>
<tr>
<td>Technology Specific</td>
<td>SecureBridge</td>
<td>The demo project demonstrates how to integrate the SecureBridge components with MyDAC to ensure secure connection to MySQL server through an SSH tunnel and SSL. This demo consists of three parts. The first part is a package that contains TMySSHIOHandler and TMySSLIOHandler component. These components provide integration with the SecureBridge library. The second part is two sample projects that demonstrate how to connect to MySQL server.</td>
</tr>
</tbody>
</table>
| Miscellaneous | 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. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CBuilder</td>
<td>A general demo project about how to create MyDAC-based applications with C++Builder. Lets you execute SQL scripts and work with result sets in a grid. This is one of the two MyDAC demos for C++Builder.</td>
</tr>
<tr>
<td>Dll</td>
<td>Demonstrates creating and loading DLLs for MyDAC-based projects. This demo project consists of two parts - an <em>My_Dll</em> project that creates a DLL of a form that sends a query to the server and displays its results, and an <em>My_Exe</em> project that can be executed to display a form for loading and running this DLL. Allows you to build a dll for one MyDAC-based project and load and test it from a separate application.</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>
</tr>
<tr>
<td>Midas</td>
<td>Demonstrates using MIDAS technology with MyDAC. This project consists of two parts: a MIDAS server that processes requests to the database and a thin MIDAS client that displays an interactive grid. This demo shows how to build thin clients that display interactive components and delegate all database interaction to a server application for processing.</td>
</tr>
<tr>
<td>VirtualTableCB</td>
<td>Demonstrates working with the <em>TVirtualTable</em> 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. This is one of the two demo projects for C++Builder</td>
</tr>
</tbody>
</table>

**MyDacDemo**

[Win32 version of the main MyDAC demo project - see above]
3.14 Deployment

MyDAC 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 MyDAC-based applications built without run-time packages, provided you are using a registered version of MyDAC.

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 MyDAC Trial Edition, you will need to deploy the following DAC BPL files and their dependencies (required IDE BPL files) with your application, even if it is built without run-time packages:

<table>
<thead>
<tr>
<th>File</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dacXX.bpl</td>
<td>always</td>
</tr>
<tr>
<td>mydacXX.bpl</td>
<td>always</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>File</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtlXX.bpl</td>
<td>always</td>
</tr>
<tr>
<td>dbrtlXX.bpl</td>
<td>always</td>
</tr>
<tr>
<td>vcldbXXX.bpl</td>
<td>always</td>
</tr>
</tbody>
</table>

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 Windows application:

<table>
<thead>
<tr>
<th>File</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dacXX.bpl</td>
<td>always</td>
</tr>
</tbody>
</table>
4 Using MyDAC

This section describes basics of using MySQL Data Access Components

- Updating Data with MyDAC Dataset Components
- Master/Detail Relationships
- Data Type Mapping
- Data Encryption
- Working in an Unstable Network
- Disconnected Mode
- Secure Connections
- Network Tunneling
- Increasing Performance
- Macros
- DataSet Manager
- Embedded Server
- National Characters
- Connection Pooling
- DBMonitor
- Writing GUI Applications with MyDAC
- Compatibility with Previous Versions
- 64-bit Development with Embarcadero RAD Studio XE2
- Database Specific Aspects of 64-bit Development
- Demo Projects
- Deployment
4.1 Updating Data with MyDAC Dataset Components

MyDAC components that are descendants from TCustomDADataset provide different means for reflecting local changes to the server.

The first approach is to use automatic generation of update SQL statements. Using this approach you should provide a SELECT statement, everything else will be made by MyDAC automatically. In case when a SELECT statement uses multiple tables, you can use UpdatingTable property to specify which table will be updated. If UpdatingTable is blank, the table, that corresponds to the first field in the dataset, is used. This approach is the most preferable and is used in most cases.

Another approach is to set update SQL statements using SQLInsert, SQLUpdate and SQLDelete properties. Set them with SQL statements which will perform corresponding data modifications on behalf of the original statement whenever insert, update or delete operation is called. This is useful when there is no possibility to generate correct statement or you need to execute some specific statements. For example, update operations should be made with stored procedure calls.

You may also assign P:Devart.MyDac.TCustomMyDataSet.UpdateObject property with the TMyUpdateSQL class instance which holds all updating SQL statements in one place. You can generate all these SQL statements using MyDAC design time editors. For more careful customization of data update operations you can use InsertObject, ModifyObject and DeleteObject properties of TMyUpdateSQL component.

See Also
- TMyQuery
- TMyStoredProc
- TMyTable
- TMyUpdateSQL

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. Lets examine how MyDAC 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.

MyDAC provides two ways to bind tables. First code example shows how to bind two TCustomMyDataSet components (TMyQuery or TMyTable) into MD relationship via parameters.

```pascal
procedure TForm1.Form1Create(Sender: TObject);
var
  Master, Detail: TMyQuery;
  MasterSource: TDataSource;
begin
  // create master dataset
  Master := TMyQuery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  Master.Open;

  Detail := TMyQuery.Create(Self);
  Detail.SQL.Text := 'SELECT * FROM Employee WHERE Dept_No = :Dept_No';
  Detail.MasterSource := MasterSource.Create(Self);
  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: TMyQuery;
begin
  Master := TMyQuery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  Master.Open;
end;
```
```delphi
Master.SQL.Text := 'SELECT * FROM Department';
// create detail dataset
Detail := TMyQuery.Create(Self);
Detail.SQL.Text := 'SELECT * FROM Employee';
// setup MD
Detail.MasterFields := 'Dept_No'; // primary key in Department
Detail.DetailFields := 'DepLink'; // foreign key in Employee
// connect detail dataset with master via TDataSource component
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
- TCustomDADataSet.Options
- TMemDataSet.CachedUpdates

© 1997-2021 Devart. All Rights Reserved.

4.3 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, MyDAC 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 MySQL database:

```sql
CREATE TABLE DECIMAL_TYPES (  ID INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  VALUE1 DECIMAL(4, 0),  VALUE2 DECIMAL(10, 0),  VALUE3 DECIMAL(15, 0),  VALUE4 DECIMAL(5, 2),  VALUE5 DECIMAL(10, 4),  VALUE6 DECIMAL(15, 6))
```

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

© 2021 Devart
To specify that numeric fields with Precision $\leq 4$ 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 := myDecimal;
  MinPrecision := 0;
  MaxPrecision := 4;
  MinScale := 0;
  MaxScale := 0;
  FieldType := ftSmallint;
  MyConnection.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
MyConnection.DataTypeMap.Clear;
// rule for DECIMAL(4,0)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, 4, 0, 0, ftSmallInt);
// rule for DECIMAL(10,0)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 10, 0, 0, 0, ftInt);
// rule for DECIMAL(5,2)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 5, 2, 0, 0, ftInt);
// rule for DECIMAL(9,1)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 9, 1, 0, 0, ftFloat);
// rule for DECIMAL(10,4)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 10, 4, 0, 0, ftBcd);
// rule for DECIMAL(15,6)
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 15, 6, 0, 0, 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 MySQL database:

```sql
CREATE TABLE DECIMAL_TYPES
(
  ID INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  VALUE1 DECIMAL(5, 2),
  VALUE2 DECIMAL(10, 4),
)```

© 2021 Devart
VALUE3 DECIMAL(15, 6)
)

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

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

If rules are set in the following way:

MyConnection.DataTypeMap.Clear;
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, 9, rlAny, rlAny, ftBCD);
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, rlAny, 0, rlAny, ftFMTBCD);

it will lead to the following result:

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

But if rules are set in the following way:

MyConnection.DataTypeMap.Clear;
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, rlAny, 0, rlAny, ftFMTBCD);
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, rlAny, 0, rlAny, ftFMTBCD);
MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 0, 9, rlAny, rlAny, ftFMTBCD);

it will lead to the following result:

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

This happens because the rule

MyConnection.DataTypeMap.AddDBTypeRule(myDecimal, 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 MySQL:

```sql
CREATE TABLE PERSON  
(  ID INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  FIRSTNAME VARCHAR(20),  LASTNAME VARCHAR(30),  GENDER_CODE VARCHAR(1),  BIRTH_DTTM DATETIME)
```

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:

```delphi
MyConnection.DataTypeMap.Clear;
MyConnection.DataTypeMap.AddDBTypeRule(myDateTime, 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:

```delphi
MyQuery.DataTypeMap.Clear;
MyQuery.DataTypeMap.AddDBTypeRule(myDateTime, 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:

```csharp
MyConnection.DataTypeMap.Clear;
MyConnection.DataTypeMap.AddDBTypeRule(myDateTime, ftDate);
MyQuery.DataTypeMap.Clear;
MyQuery.DataTypeMap.AddDBTypeRule(myDateTime, ftDateTime);
```

In this case, in all DataSets for the DATETIME type fields with the ftDate type will be created, and for MyQuery - 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:

```sql
CREATE TABLE ITEM  
(
    ID INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    NAME CHAR(50),
    GUID CHAR(38)
)
```

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:

```csharp
MyQuery.DataTypeMap.Clear;
MyQuery.DataTypeMap.AddDBTypeRule(myChar, 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:

```csharp
MyQuery.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:

```
MyConnection.DataTypeMap.AddDBTypeRule(myVarchar, 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.

<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.
4.4 Data Encryption

MyDAC 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.

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.IndexFieldNames 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 INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  ENAME VARBINARY(2000) DEFAULT NULL,  HIREDATE VARBINARY(200) DEFAULT NULL,  SAL VARBINARY(200) DEFAULT NULL,  FOTO BLOB DEFAULT NULL)
```

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.

```sql
MyQuery.SQL.Text := 'SELECT * FROM EMP';
MyQuery.Encryption.Encryptor := MyEncryptor;
MyQuery.Encryption.Fields := 'ENAME, HIREDATE, SAL, FOTO';
MyEncryptor.Password := '11111';
MyQuery.DataTypeMap.AddFieldNameRule ('ENAME', ftString);
MyQuery.DataTypeMap.AddFieldNameRule ('HIREDATE', ftDateTime);
MyQuery.DataTypeMap.AddFieldNameRule ('SAL', ftFloat);
MyQuery.Open;
```
4.5 Working in an Unstable Network

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

```pascal
TCustomDAConnection.Options.LocalFailover = True
TCustomDAConnection.Options.DisconnectedMode = True
TDataSet.CachedUpdates = True
TCustomDADataset.FetchAll = True
TCustomDADataset.Options.LocalMasterDetail = 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), MyDAC can implicitly perform the following operations:

```pascal
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.
4.6 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.

See Also
- TCustomDAConnection.Options
- FetchAll
- Devart.MyDac.TMyQuery.LockMode
- TCustomDAConnection.Pooling
- TCustomDAConnection.Connect
- TCustomDAConnection.Disconnect
- Working in unstable network

4.7 Secure Connections

This section describes the ways to secure connection between a client and a MySQL server.
- Connecting via SSL
- Connecting via SSH
- HTTP/HTTPS Network Tunneling

4.7.1 Connecting via SSL

Connecting to MySQL 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 MySQL server from a Delphi application. MySQL 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 MySQL server. This
tutorial demonstrates how to create a sample Delphi application that connects to PostgreSQL using SSL as the encryption method.

Before connecting to MySQL via SSL, create SSL certificates as explained in the MySQL documentation and configure SSL parameters in the my.ini file.

Sample Delphi App That Connects to MySQL Using SSL

To create an SSL connection to MySQL, set the T:Devart.MyDac.TMyConnectionOptions.Protocol. property in Options to mpSSL. It forces the application to only connect through SSL - if a connection attempt fails, an exception is raised.

MyConnection.Options.Protocol := mpSSL;

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

1. Run RAD Studio and select 'File -> New – > VCL Forms Application – Delphi'.
2. Place the TCRSSLIOHandler component, which allows MyDAC to connect to MySQL server through SSL, onto the form. Also add the TMyConnection, TMyQuery, TDataSource, TDBGrid, and TButton components to the form - they are required to create a sample application that connects to the MySQL server via SSL, runs a selection operation against the database, and displays the obtained rows in a data grid.
3. Select the **TMyConnection** component and assign the **TCRSSLIOHandler** object to the **IOHandler** property in the Object Inspector.

4. Expand **Options** and set the **Protocol** property to **mpSSL**.

5. 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.

6. Double-click **TMyConnection** and specify the server address, port, username, password, and, optionally, database name. Click **Connect** to test connection to the MySQL server.

7. Select the **TDataSource** component and assign the **MyQuery1** object to the **DataSet** property.

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

10. Select the **TButton** component and create the **OnClick** event. Add the code that will call the **Open** method in the **TMyQuery** component when you click the button.

```pascal
procedure TForm1.Button1Click(Sender: TObject);
begin
  MyQuery1.Open;
end;
```

11. 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 MySQL in Delphi Using the OpenSSL Library

Another way to embed SSL client functionality into your Delphi app, which uses MyDAC 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 MySQL Through 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 the MySQL server resides. The SSH client connects to the SSH server, which sends all commands to the remote MySQL server. This tutorial demonstrates how to create a sample Delphi application that connects to MySQL 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 MySQL server. You can build the SSH server demo project that is distributed with SecureBridge ('Documents\Devart\MyDAC for RAD Studio\Demos\TechnologySpecific\SecureBridge\Demo') and run the executable file.

After installing MyDAC and SecureBridge software on your system, install the TCRSSHIOHandler component in RAD Studio to bind MyDAC with SecureBridge. The installation instructions are provided in the Readme.html, which is located by default in "My Documents\Devart\MyDAC for RAD Studio XX\Demos\TechnologySpecific\SecureBridge".
Sample Delphi App that Connects to MySQL Using SSH

1. Run RAD Studio and select 'File -> New -> VCL Forms Application - Delphi'.
2. Place the following components on the form: TCRSSHIOHandler, TMyConnection, TMyQuery, TScFileStorage, TScSSHClient, TDataSource, TMyQuery, TDBGrid, and TButton. The sample application will connect to the MySQL 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 **TMyQuery** and set the **Connection** property to **MyConnection1**. Double-click the component and enter a SQL statement to be executed against the MySQL database.
6. Double-click the TButton to switch to the unit view. Add the code to call the Open method on the MyQuery1 object to activate the dataset when the button is clicked.

```pascal
procedure TForm3.Button1Click(Sender: TObject);
begin
  MyQuery1.Open;
end;
end.
```

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 TMyConnection component. Specify your server address, port, database name (optionally), and username and password for the MySQL user. Set the IOHandler property to CRSSHIOHandler1. Click Connect to check connection to the MySQL server.

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.
2. SSH Using OpenSSH or Any Other Third-Party SSH Tunnel

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

The following is the step-by-step sequence of actions for the easiest case of using OpenSSH for Windows. The detailed description of each command you can see in the documentation for OpenSSH.


2. Install SSH server
   - Choose a machine that will be used as SSH server. It does not have to be the same machine that is a MySQL server, but communication channel between SSH server and MySQL server must be protected
   - Using Windows Control Panel create a user and set a password for him. For example, SSHUser with password SSHPass
   - Install Open SSH. It is enough to install only Server components
   - Open OpenSSH/bin folder
   - Add SSHUser to the list of allowed users: `mkpasswd -l -u SSHUser >> ..\etc\passwd`
   - Use mkgroup to create a group permissions file `mkgroup -l >> ..\etc\group`
   - Run OpenSSH service `net start opensshd`

3. Install SSH client
   - Choose a machine that will be used as SSH client. It does not have to be the same
machine where client application (MySQL client) is running, but communication channel between SSH client and MySQL client must be protected

- Install Open SSH to SSH client. You may not install server components
- Run SSH client

```
ssh.exe -L <SSH port>:<MySQL server>:<MySQL server port> <SSHUser>@<SSH server>
```

- `<SSH port>` - port number of SSH client that will be redirected to the corresponding port of MySQL server
- `<MySQL server>` - name or IP address of the machine where MySQL server is installed
- `<MySQL server port>` - number of MySQL server port. As usual, 3306.
- `<SSHUser>` - user name created in p. 2
- `<SSH server>` - name or IP address of the machine where SSH server is installed in p. 2

For example,

```
ssh.exe -L 3307:server:3306 SSHUser@192.168.0.116
```

At the first start you will be suggested to confirm a connection with the specified SSH server. Enter "yes" for confirmation.

On each start of SSH you must enter a password set in p. 2

4. Configure TMyConnection

```
MyConnection1.Server := <SSH client>;
MyConnection1.Port := <SSH port>;
```

If SSH client was installed at the same machine as MySQL client, you can assign 'localhost' to MyConnection1.Server.

Pay attention that in the specified sequence above check of SSHUser authentication is performed by Windows. About the methods of higher protection (key authentication etc) see documentation for OpenSSH.

To get more detailed information on using encrypted connections refer to MySQL Reference Manual.
necessary to use a proxy or bypass a firewall. This article describes different ways to connect to MySQL server with MyDAC.

- **Direct connection**
- **Connection through HTTP tunnel**
  - Connection Through proxy and HTTP tunnel
- Sample Delphi App That Connects to MySQL 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:

```delphi
MyConnection := TMyConnection.Create(self);
MyConnection.Server := 'localhost';
MyConnection.Port := 3306;
MyConnection.Username := 'root';
MyConnection.Password := 'root';
MyConnection.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 MyDAC together with HTTP tunneling software to connect to MySQL server. MyDAC 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 to use HTTP tunneling. The script must be available 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\MyDac for Delphi X\HTTP\tunnel.php. The only requirement to the server is PHP 5 support.

To connect to the database, you should set TMyConnection parameters for usual direct connection, which will be established from the web server side, the Options.Protocol property to mpHttp, 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</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>
<tr>
<td>HttpOptions.Password</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Connection Through Proxy and HTTP Tunnel

HTTP tunneling server is not directly accessible from client machine. For example, client address is 10.0.0.2, server address is 192.168.0.10, and the MySQL server listens on port 3307. 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 TMyConnection.HttpOptions options you have to setup a HttpOptions.ProxyOptions object as follows:

```delphi
MyConnection := TMyConnection.Create(self);
MyConnection.Server := '192.168.0.10';
MyConnection.Port := 3307;
MyConnection.Username := 'root';
MyConnection.Password := 'root';
MyConnection.Options.Protocol := mpHttp;
MyConnection.HttpOptions.ProxyOptions.Hostname := '10.0.0.1';
MyConnection.HttpOptions.ProxyOptions.Port := 808;
MyConnection.Connect;
```

Note that setting parameters of MyConnection.HttpOptions.ProxyOptions automatically enables proxy server usage.

Sample Delphi App That Connects to MySQL 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: TMyConnection, TMyQuery, TDataSource, TDBGrid, TButton, and TCRSSLIOHandler. The last component is required when you connect through HTTPS. Note that the TCRSSLIOHandler is distributed with SecureBridge and is required to bind MyDAC with SecureBridge. The installation instructions for the component are provided in the Readme.html, which is located by default in "My Documents\Devart\MyDAC for RAD Studio\Demos\TechnologySpecific\SecureBridge\DelphiXX".

Select TDBGrid and set the DataSource property to DataSource1.
In the **TDataSource** component, assign **TMyQuery1** to the **DataSet** property.

Select **TMyQuery** and set the **Connection** property to **MyConnection1**. Double-click the component and enter an SQL statement to be executed against the MySQL database.
• Double-click **TButton** to switch to the unit view. Add code that invokes the **Open** method on the **MyQuery1** object to activate the dataset when the button is clicked.

```delphi
procedure TForm1.Button1Click(Sender: TObject);
begin
    MyQuery1.Open;
end;
```

• Select the **TMyConnection** component. If you use the secure tunnel, set the **IOHandler** property to **CRISSLIOHandler1**. Expand **HttpOptions** and enter the URL of the tunnel.php script on your server.
• In `TMyConnection`, expand Options and set the Protocol property to `mpHttp`.

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

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

4.8 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 TMyLoader. Unfortunately, TMyLoader 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 TMyLoader, 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 INT,
  F_INTEGER INT,
  F_FLOAT FLOAT,
  F_STRING VARCHAR(250),
  F_DATE DATETIME,
  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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<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:

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

```pascal
var
  i: Integer;
begin
  // describe the SQL query
  MyQuery1.SQL.Text := 'INSERT INTO BATCH_TEST VALUES (:ID, :F_INTEGER, ... := Now;
  // define the parameter types passed to the query:
  MyQuery1.Params[0].DataType := ftInteger;
  MyQuery1.Params[1].DataType := ftInteger;
  MyQuery1.Params[2].DataType := ftFloat;
  MyQuery1.Params[3].DataType := ftString;
  MyQuery1.Params[4].DataType := ftDateTime;
  // specify the array dimension:
  MyQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to MyQuery1.Params.ValueCount - 1 do begin
    MyQuery1.Params[0][i].AsInteger := i + 1;
    MyQuery1.Params[1][i].AsInteger := i + 2000 + 1;
    MyQuery1.Params[2][i].AsFloat := (i + 1) / 12;
    MyQuery1.Params[3][i].AsString := 'Values ' + IntToStr(i + 1);
    MyQuery1.Params[4][i].AsDateTime := Now;
  end;
  // insert 1000 rows into the BATCH_TEST table
  MyQuery1.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:

```
MyQuery1.Execute(1000);
```

2×500 rows:

```
// insert first 500 rows
MyQuery1.Execute(500, 0);
// insert next 500 rows
MyQuery1.Execute(500, 500);
```

500 rows, then 300, and finally 200:

```
// insert 500 rows
MyQuery1.Execute(500, 0);
// insert next 300 rows starting from 500
MyQuery1.Execute(300, 500);
```
// insert next 200 rows starting from 800
MyQuery1.Execute(200, 800);

**Batch UPDATE operation sample**

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

```pascal
var
  i: Integer;
begin
  // describe the SQL query
  // define parameter types passed to the query:
  MyQuery1.Params[0].DataType := ftInteger;
  MyQuery1.Params[1].DataType := ftFloat;
  MyQuery1.Params[2].DataType := ftString;
  MyQuery1.Params[3].DataType := ftDateTime;
  MyQuery1.Params[4].DataType := ftInteger;
  // specify the array dimension:
  MyQuery1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to 1000 - 1 do begin
    MyQuery1.Params[0][i].AsInteger := i - 2000 + 1;
    MyQuery1.Params[1][i].AsFloat := (i + 1) / 100;
    MyQuery1.Params[2][i].AsString := 'New Values ' + IntToStr(i + 1);
    MyQuery1.Params[3][i].AsDateTime := Now;
    MyQuery1.Params[4][i].AsInteger := i + 1;
  end;
  // update 1000 rows in the BATCH_TEST table
  MyQuery1.Execute(1000);
end;
```

**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
  MyQuery1.SQL.Text := 'DELETE FROM BATCH_TEST WHERE ID=:ID';
  // define parameter types passed to the query:
  MyQuery1.Params[0].DataType := ftInteger;
  // specify the array dimension
  MyQuery1.Params.ValueCount := 1000;
  // populate the arrays with parameter values
  for i := 0 to 1000 - 1 do
    MyQuery1.Params[0][i].AsInteger := i + 1;
  // delete 1000 rows from the BATCH_TEST table
  MyQuery1.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>
<th>Standard Operation (sec.)</th>
<th>Batch Operation (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>1138</td>
<td>11.02</td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td>1637</td>
<td>26.72</td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>1444</td>
<td>17.66</td>
<td></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:

```delphi
for i := 0 to 9999 do begin
  MyQuery1.Params.ParamByName('ID')[i].AsInteger := i + 1;
  MyQuery1.Params.ParamByName('F_INTEGER')[i].AsInteger := i + 2000 + 1;
  MyQuery1.Params.ParamByName('F_FLOAT')[i].AsFloat := (i + 1) / 12;
  MyQuery1.Params.ParamByName('F_STRING')[i].AsString := 'Values ' + IntToStr(i + 1);
  MyQuery1.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.

4.9 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 (TCustomDACConnection.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 TDatasetOptions.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 the TMyCommand component is more preferable than TMyQuery. 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 TDatasetOptions.CompressBlobMode. You can also use TMyConnection.Options.Compress. Setting TMyTable.Options.UseHandler can give an additional performance under high server load.

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 `TDDevart.MyDac.TMyLoader` component instead of Insert/Post methods, or execution of the `INSERT` commands multiple times in a cycle. Sometimes usage of `TDDevart.MyDac.TMyLoader` improves performance several times.

© 1997-2021 Devart. All Rights Reserved.

### 4.10 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 MyDAC 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:

```
SELECT * FROM &TableName
```

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

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

MyDAC 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 `TMacroAsString` 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.

```
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. For example, the following is a script that creates a new user account and grants required privileges.

```
Script1.SQL.Add('CREATE USER &Username IDENTIFIED BY &Password;');
Script1.SQL.Add('GRANT &Privileges TO &Username;');
```

To execute the script for another user you do not have to change the script SQL property, you can just set required 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.

Request Support        DAC Forum        Provide Feedback

4.11  **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.
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.

![DataSet Manager Window](image)

**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.

![DataSet Manager - Options](image)

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.

© 1997-2021 Devart. All Rights Reserved.

4.12 Embedded Server

Since version 4.0 MySQL server supports Embedded server. Embedded server is an easy to install server used by applications that do not require multi-user work with MySQL server. For example, Embedded server can be used for money access machines, automatic cash desks, different electronic facilities and so on. Please refer to MySQL Reference Manual for more details on features and using of Embedded server. Also you can find some information about licensing Embedded server in MySQL Reference Manual. Please refer to Embedded...
Demo for a sample.

**Which version of Embedded server to use**

MySQL Embedded Server 4.0 should be recompiled to be used in your application.

MySQL Embedded Sever 5.0 is not included into the binary installation pack. Below is a quotation from the [MySQL Reference Manual](#):

"The Embedded MySQL server library is NOT part of MySQL 5.0. It is part of previous editions and will be included in future versions, starting with MySQL 5.1."

That is why we have not tested MyDAC with the MySQL Embedded Sever 5.0.

So, we recommend using MySQL Embedded Server 4.1. As MySQL Embedded Server has some problems working with the InnoDB storage, we recommend disabling this storage engine. You can do this by checking the "Disable InnoDB storage engine" option in the TMyEmbConnection editor on the Params tab. Another way is to add the --skip-innodb parameter to the TMyEmbConnection component manually.

**Installation**

**Windows**

Copy libmysqld.dll file to the folder available for executable file of the application. Please see a detailed description of accessible paths at LoadLibrary description

A typical structure of folders for an application using Embedded Server:

- **Project.exe** - executable file of your application
- **libmysqld.dll** - MySQL Embedded server library
- **share/english/errmsg.sys** - file with MySQL Embedded server messages
- **data/** - data directory (DataDir). See a structure of this folder in MySQL Reference Manual
- **data/mysql/** - directory with service data of MySQL (user access rights, and so on)
- **DataBase/** - directory with user data. See [TCustomMyConnection.DataBase](#)

**Linux**

- Copy libmysqld.so.14.0.0 file to /usr/lib folder
- At /usr/lib folder execute the following commands to create links:

  ln libmysqld.so.14.0.0 libmysqld.so
  ln libmysqld.so.14.0.0 libmysqld.so.14

- Copy files needed for working of Embedded Server. As a rule, it is errormessage file, for example share/english/errmsg.sys.
- Create a folder for data
• If it is necessary, copy files with data to the data folder

Settings

On the start (first opening a connection), MySQL Embedded Server searches for the setting values in the next order:
• TMyEmbConnection.Params
• [<Application exe-file name (with extension)>] section of configuration file (my.ini or my.cnf) - settings specific to particular application.
• [Embedded] section of configuration file - settings specific to Embedded server
• [Server] section of configuration file - common settings for MySQL server and Embedded server.

Usually to set-up Embedded Server it is enough to set basedir and datadir. But sometimes some additional settings are required, for example to disable usingInnoDB engine (--skip-innodb). The detailed list of settings you can find at MySQL Reference Manual.

Pay attention that all paths must be set through "/" but not "\".

Note, parameters names are case-sensitive.

If datadir is located in the read-only storage, then you need to set OnLog and OnLogError event handlers to prevent server from attempts to create log-files in datadir.

Limitations

Simultaneous access to the same data from several instances of MySQL server (for example, to MySQL server and Embedded server) can be a reason of data loss.

See Also

• Embedded Demo
• TMyEmbConnection
• TMyEmbConnection.Params
• TMyConnection.Options.Embedded

© 1997-2021
Devart. All Rights Reserved.

4.13 National Characters

When data is transferred data between the client and server, the server needs to know the
encoding used by the client. You can set the encoding by configuring the MySQL server settings (see the MySQL Reference Manual for details), or by using the TMyConnection.Options_CHARSET or TMyConnection.Options.UseUnicode properties on the client. The former way is less suitable as it requires access to the server settings, which is not always possible. The latter way is more convenient, but it may cause some delays.

The CHARSET and UseUnicode properties are mutually exclusive, thus when you set UseUnicode to True, the value of CHARSET is ignored. By default, CHARSET = '' and UseUnicode = False. The server makes conversions according to its settings.

If a character set is assigned to the CHARSET property, the SET NAMES <CHARSET> statement is passed to the server when establishing a connection, to notify it about the character used by the client. To get a list of available charsets, execute the SHOW CHARSET statement. Note that when you assign utf8 to this property, the values of all string fields are converted to UTF-8, which may prevent you from using data-aware components.

If you need to receive string data in the Unicode format on the client to work with almost any language, or you connect with a username or password that contains characters like £, set the UseUnicode property to True. All TStringField values will be converted to TWideStringField. This behaviour is useful when, for example, you create a database of books in the library, and you need to store the original title of the book alongside the translated title. The downside is that all string data is converted on the client, which may cause some delays.

See Also
- TCustomMyConnection.Options
- TMyConnection.Options

4.14 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 TCustomDAConnection component to True. Also you should set the PoolingOptions of the TCustomDAConnection. 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, Server, T:Devart.Odac.TOraSession, Password, Database, IsolationLevel, Port, IOHandler, ConnectionTimeout, Compress, Direct, Embedded, Protocol, Charset, UseUnicode, NumericType. 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 TCustomDAConnection component when it connects to the database. The pool validates connection when it is placed to the pool (e.g. when the TCustomDAConnection 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 TCustomDAConnection 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 TCustomDAConnection 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 TCustomDACConnection 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 TCustomDACConnection component disconnects, the RemoveFromPool method of TCustomDACConnection can be used. You can also free all connection in the pool by using the class procedures Clear or AsyncClear of TMyConnectionPoolManager. 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 TCustomDACConnection component set to True. In this case internal connections can be shared between TCustomDACConnection components. When some operation is performed on the TCustomDACConnection 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 TCustomDACConnection component it can use the same connection from the pool.

See Also
- TCustomDACConnection.Pooling
- TCustomDACConnection.PoolingOptions
- Working with Disconnected Mode

4.15 DBMonitor

To extend monitoring capabilities of MyDAC applications there is an additional tool called DBMonitor. It is provided as an alternative to Borland SQL Monitor which is also supported by MyDAC.

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 `TMySQLMonitor` 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.

4.16 Writing GUI Applications with MyDAC

MyDAC 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 MyDacVcl unit in your application. This feature is needed for writing console applications.

*Delphi and C++Builder*

By default MyDAC does not require Forms, Controls and other GUI related units. Only `TMyConnectDialog` component require the Forms unit.

4.17 Compatibility with Previous Versions

We always try to keep MyDAC compatible with previous versions, but sometimes we have to change the behaviour of MyDAC in order to enhance its functionality, or avoid bugs. This topic describes such changes, and how to revert the old MyDAC behaviour. We strongly recommend not to turn on the old behaviour of MyDAC. Use options described below only if changes applied to MyDAC 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 MyDAC 4.00.2.8. To restore old behavior, set the BaseSQLOldBehavior variable to True.

**DBAccess.SQLGeneratorCompatibility:**

If the manually assigned **RefreshSQL** property contains only "WHERE" clause, MyDAC 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 MyDAC 5.00.0.4. To restore the old behavior, set the BaseSQLOldBehavior variable to True.

**MemDS.SendDataSetChangeEventAfterOpen:**

Starting with MyDAC 5.20.0.11, 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.DoNotRaiseExcetionOnUaFail:**

Starting with MyDAC 5.20.0.12, 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 DoNotRaiseExcetionOnUaFail to True.

**MyClasses.__Strings65535ToMemo:**

Control flow functions of MySQL (like IF, CASE) change data type of LONGMEMO and LONGBLOB fields. It causes wrong description of these fields by MyDAC and truncating their data. To avoid these problems, MyDAC tries to restore the correct data type. This behaviour was introduced in MyDAC 5.10.0.9. To disable this behaviour, set the __Strings65535ToMemo variable to False.

**DBAccess.ParamStringAsAnsiString:**

This variable has sense for Delphi 2009 and higher.

Set its value to True to use the AsAnsiString property when setting the parameter value through TDAParam.AsString. Otherwise the AsWideString property is used. The default value is False.
DBAccess.RefreshParamsOnInsert:

Starting with MyDAC 5.50.0.36, when master/detail relationship is used on inserting a new record into master table parameters in detail table are not updated. To restore the old behavior, set the RefreshParamsOnInsert variable to True.

© 1997-2021 Devart. All Rights Reserved.

4.18 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:
That is why, the following code will work incorrectly on the 64-bit platform:

\[ \text{Ptr} := \text{Pointer} (\text{Integer} (\text{Ptr}) + \text{Offset}) ; \]

While this code will correctly on the 64-bit platform and incorrectly on the 32-bit platform:

\[ \text{Ptr} := \text{Pointer} (\text{Int64} (\text{Ptr}) + \text{Offset}) ; \]

For this purpose, the following platform-dependent integer type is introduced:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>NativeInt</td>
<td>4 bytes</td>
<td>8 bytes</td>
</tr>
<tr>
<td>NativeUInt</td>
<td>4 bytes</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

This type helps ensure that pointers work correctly both for the 32- and 64-bit platforms:

\[ \text{Ptr} := \text{Pointer} (\text{NativeInt} (\text{Ptr}) + \text{Offset}) ; \]

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>TYPE</th>
<th>Delphi Version</th>
<th>Size</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:

```pascal
SendMessage(hWnd, WM_SETTEXT, 0, LPARAM(@MyCharArray));
```

Wrong:

```pascal
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:

```pascal
SetWindowLongPtr(hWnd, GWLP_WNDPROC, LONG_PTR(@MyWindowProc));
```

Wrong:

```pascal
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:

```pascal
Message.Result := LRESULT(Self);
```
Wrong:

```pascal
Message.Result := Integer(Self);
```

All TWM...-records for the windows message handlers must use the correct Windows types for the fields:

```pascal
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:

- 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. You 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)
4.19 Database Specific Aspects of 64-bit Development

MySQL Connectivity Aspects

Client mode:

If you are developing a 64-bit application, you have to be aware of specifics of working with client libraries at design-time and run-time. To connect to a MySQL database at design-time, you must have its 32-bit client library. You have to place it to the C:\Windows\SysWOW64 directory. This requirement flows out from the fact that RAD Studio XE2 is a 32-bit application and it cannot load 64-bit libraries at design-time. To work with a MySQL database in run-time (64-bit application), you must have the 64-bit client library placed to the C:\Windows\System32 directory.

DIRECT mode:

Since there is no need to install client library for the DIRECT mode, the specifics of developing applications that use MyDAC as data access components, depends exclusively on peculiarities of each target platform.

5 Reference

This page shortly describes units that exist in MyDAC.

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>Package</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DAAlerter</td>
<td>This unit contains the base class for the TMyAlerter component.</td>
</tr>
<tr>
<td>DADump</td>
<td>This unit contains the base class for the TMyDump component.</td>
</tr>
<tr>
<td>DALoader</td>
<td>This unit contains the base class for the TMyLoader component.</td>
</tr>
<tr>
<td>DAScript</td>
<td>This unit contains the base class for the TMyScript component.</td>
</tr>
<tr>
<td>DASQLMonitor</td>
<td>This unit contains the base class for the TMySQLMonitor 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.MyDac.DataAdapter</td>
<td>This unit contains implementation of the MyDataAdapter class.</td>
</tr>
<tr>
<td>MemData</td>
<td>This unit contains classes for storing data in memory.</td>
</tr>
<tr>
<td>MemDS</td>
<td>This unit contains implementation of the TMemDataSet class.</td>
</tr>
<tr>
<td>MyAccess</td>
<td>This unit contains implementation of most public classes of MyDAC.</td>
</tr>
<tr>
<td>MyBackup</td>
<td>This unit contains implementation of the TMyBackup component.</td>
</tr>
<tr>
<td>MyBuilderClient</td>
<td>This unit contains implementation of the MyBuilder class.</td>
</tr>
<tr>
<td>MyClasses</td>
<td>This unit contains implementation of the EMyError class.</td>
</tr>
<tr>
<td>MyConnectionPool</td>
<td>This unit contains the TMyConnectionPoolManager class for managing</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MyDacVcl</td>
<td>This unit contains the visual constituent of MyDAC.</td>
</tr>
<tr>
<td>MyDump</td>
<td>This unit contains implementation of the TMyDump component.</td>
</tr>
<tr>
<td>MyEmbConnection</td>
<td>This unit contains implementation of the TMyEmbConnection component.</td>
</tr>
<tr>
<td>MyLoader</td>
<td>This unit contains implementation of the TMyLoader component.</td>
</tr>
<tr>
<td>MyScript</td>
<td>This unit contains implementation of the TMyScript component.</td>
</tr>
<tr>
<td>MyServerControl</td>
<td>This unit contains implementation of the TMyServerControl component.</td>
</tr>
<tr>
<td>MySqlApi</td>
<td>This unit contains implementation of the class.</td>
</tr>
<tr>
<td>MySQLMonitor</td>
<td>This unit contains implementation of the TMySQLMonitor component.</td>
</tr>
<tr>
<td>MySqlVioPipe</td>
<td>This unit contains implementation of the TCRIOHandler class.</td>
</tr>
<tr>
<td>VirtualDataSet</td>
<td>This unit contains implementation of the TVirtualDataSet component.</td>
</tr>
<tr>
<td>VirtualTable</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
### TCRCursor

A base class for classes that work with database cursors.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBeforeFetchProc</td>
<td>This type is used for the TCustomDAODataSet.BeforeFetch 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>

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
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

© 1997-2021 Devart. All Rights Reserved.

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 (inherited from TSharedObject)</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 (inherited from TSharedObject)</td>
<td>Increments the reference count for the number of references dependent on the TSharedObject object.</td>
</tr>
</tbody>
</table>
### 5.1.2 Types

Types in the **CRAccess** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBeforeFetchProc</td>
<td>This type is used for the TCustomDADataset.BeforeFetch event.</td>
</tr>
</tbody>
</table>

#### 5.1.2.1 TBeforeFetchProc Procedure Reference

This type is used for the TCustomDADataset.BeforeFetch event.

**Unit**

*CRAccess*

**Syntax**

```
TBeforeFetchProc = procedure (var Cancel: boolean) of object;
```

**Parameters**

**Cancel**

True, if the current fetch operation should be aborted.

### 5.1.3 Enumerations

Enumerations in the **CRAccess** unit.
## 5.1.3.1 TCRIsolationLevel Enumeration

Specifies how to handle transactions containing database modifications.

### Unit

**CRAccess**

### Syntax

```csharp
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>ilIsolated</td>
<td>The most restricted level of transaction isolation. Database server isolates data involved in current transaction by putting 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.</td>
</tr>
<tr>
<td>ilReadCommitted</td>
<td>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.</td>
</tr>
<tr>
<td>ilReadUnCommitted</td>
<td>The most unrestricted level of the transaction isolation. All types</td>
</tr>
</tbody>
</table>
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**

```
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

```cpp
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>
<tbody>
<tr>
<td>TCRBatchMove</td>
<td>Transfers records between datasets.</td>
</tr>
</tbody>
</table>

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.OnBatchMo</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

\[ \text{TCRBatchMove} = \textbf{class}(\text{TComponent}); \]

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 \text{TCRBatchMove.Mode} property determines the desired operation type, the \text{TCRBatchMove.Source} and \text{TCRBatchMove.Destination} properties indicate corresponding datasets.

\textbf{Note:} A TCRBatchMove component is added to the Data Access page of the component palette, not to the MyDAC page.

5.2.1.1.1 Members

\textbf{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>ChangedCount</td>
<td>Used to get the number of records changed in the destination dataset.</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>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------</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>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>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 added to the destination dataset because of the field type mismatch.</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 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 added to the destination dataset because of the field type mismatch.</td>
</tr>
</tbody>
</table>

Published
<table>
<thead>
<tr>
<th>Field</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 dataset for the batch operation.</td>
</tr>
</tbody>
</table>

See Also
- TCRBatchMove Class
- TCRBatchMove Class Members

© 1997-2021  Request Support   DAC Forum   Provide Feedback
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

```
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

```
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.
5.2.1.1.2.3 ChangedCount Property

Used to get the number of records changed in the destination dataset.

Class

TCRBatchMove

Syntax

```pascal
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.

© 1997-2021 Devart. All Rights Reserved.

5.2.1.1.2.4 CommitCount Property

Used to set the number of records to be batch moved before commit occurs.

Class

TCRBatchMove

Syntax

```pascal
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 Kb.

© 1997-2021 Devart. All Rights Reserved.
5.2.1.2.5 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.

5.2.1.2.6 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.

5.2.1.2.7 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

```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 `AbortOnKeyViol` is True, then KeyViolCount will never exceed one, because the operation aborts when the integrity or key violation occurs.

See Also

- `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

```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:

ColName

Example
To map a column named SourceColName in the source dataset to the column named DestColName in the destination dataset, use:

```
DestColName=SourceColName
```

5.2.1.1.2.9 Mode Property

Used to set the type of the batch operation that will be executed after calling the `Execute` method.

**Class**

`TCRBatchMove`

**Syntax**

```
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.

5.2.1.1.2.10 MovedCount Property

Used to get the number of records that were read from the source dataset during the batch operation.

**Class**

`TCRBatchMove`

**Syntax**

```
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.

5.2.1.1.2.11 ProblemCount Property

Used to get the number of records that could not be added to the destination dataset because of the field type mismatch.

Class

TCRBatchMove

Syntax

```pascal
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

5.2.1.1.2.12 RecordCount Property

Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.

Class

TCRBatchMove

Syntax

```pascal
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.

5.2.1.1.3 Methods

Methods 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>Execute</td>
<td>Performs the batch operation.</td>
</tr>
</tbody>
</table>
5.2.1.1.3.1 Execute Method

Performs the batch operation.

Class

TCRBatchMove

Syntax

```plaintext
procedure Execute;
```

Remarks

Call the Execute method to perform the batch operation.

<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>

See Also

• TCRBatchMove Class

• TCRBatchMove Class Members
5.2.1.1.4.1 OnBatchMoveProgress Event

Occurs when providing feedback to the user about the batch operation in progress is needed.

Class

TCRBatchMove

Syntax

property OnBatchMoveProgress: TCRBatchMoveProgressEvent;

Remarks

Write the OnBatchMoveProgress event handler to provide feedback to the user about the batch operation progress.

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

```cpp
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.

<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.
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>

---

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

```
TCREncryptor = class(TComponent);
```

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>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>

See Also
- TCREncryptor Class
- TCREncryptor Class Members
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`.

© 1997-2021 Devart. All Rights Reserved.

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 `eaBlowfish`.

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

**Syntax**

```pascal
property HashAlgorithm: TCRHashAlgorithm default haSHA1;
```

**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 `DataHeader` property is set to `ehTagAndHash`). Default value is `haSHA1`.

**InvalidHashAction Property**

Specifies the action to perform on data fetching when hash data is invalid.

**Class**

`TCREncryptor`

**Syntax**

```pascal
property InvalidHashAction: TCRInvalidHashAction default ihFail;
```

**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 `DataHeader` property is set to `ehTagAndHash`). Default value is `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
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.

5.3.1.3 Methods

Methods of the TCREncryptor class.

For a complete list of the TCREncryptor class members, see the TCREncryptor Members topic.

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.

© 1997-2021 Devart. All Rights Reserved.

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</td>
</tr>
</tbody>
</table>
5.3.2.1 TCREncDataHeader Enumeration

Specifies whether the additional information is stored with the encrypted data.

Unit
CREncryption

Syntax
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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>haMD5</td>
<td>The MD5 hash algorithm is used.</td>
</tr>
<tr>
<td>haSHA1</td>
<td>The SHA-1 hash algorithm is used.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback
5.3.2.4 TCRInvalidHashAction Enumeration

Specifies the action to perform on data fetching when hash data is invalid.

Unit

CXEncryption

Syntax

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.
### TCRDBGrid Class

Extends the functionality of the TDBGrid component.

For a list of all members of this type, see TCRDBGrid members.

**Unit**

`CRGrid`

**Syntax**

```plaintext
TCRDBGrid = class(TCustomDBGrid);
```

### Members

**TCRDBGrid** class overview.

**Properties**

<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>
Set `LevelDelimiterChar` to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.

Write an `OnMemoClick` event handler to provide custom processing of Memo fields instead of built-in Memo field editor.

OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property.

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 <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 Column property.</td>
</tr>
<tr>
<td><strong>ActivateSearchEdit</strong></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 Column property.</td>
</tr>
<tr>
<td><strong>AdjustColumns</strong></td>
<td>Call <code>AdjustColumns</code> method at run-time to stretch all columns in a grid so that</td>
</tr>
</tbody>
</table>
they become wide enough to accommodate every visible field value. Widths of column captions restrict minimum column sizes for this operation.

**ApplyFilter**

Call ApplyFilter method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.

**CalcTitleLevel**

Call CalcTitleLevel method to set Top and Bottom values of aRect parameter record depending on Level parameter value.

**ClearFilters**

Call CalcFilters method to clear all filter statements for grid columns.

**ClearSorting**

Call ClearSorting method to discard sorting previously applied to the grid data.

**DataChanged**

Call DataChanged method to update displayed data and status information for the grid.

**GetGridSize**

Call GetGridSize method to obtain width of a client area for the grid component.

**GetTitleLevel**

GetTitleLevel method returns TRect structure filled with top and bottom coordinates for the specified title level.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnGetCellParams</strong></td>
<td>Write an OnGetCellParams event handler to provide custom background colors while the grid redraws individual cells.</td>
</tr>
</tbody>
</table>
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>OptionsMenu</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>Columns</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>Filtered</td>
<td>Set Filtered property to specify whether record entries are filtered according to the conditional statements of individual columns or not.</td>
</tr>
<tr>
<td>LevelDelimiterChar</td>
<td>Set LevelDelimiterChar to set the character symbol which TCRDBGrid looks for in column captions to resolve multilevel nested columns.</td>
</tr>
<tr>
<td>OnMemoClick</td>
<td>Write an OnMemoClick event handler to provide</td>
</tr>
</tbody>
</table>
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

```delphi
property Columns: TCRDBGridColumns stored False;
```

Remarks

When browsing TDBGridColumns remember that its Items property actually returns TCRColumn objects when indexed.

5.4.1.1.2.2 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.

5.4.1.1.2.3  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 '|';
```

5.4.1.1.2.4  OnMemoClick Property

Write an OnMemoClick event handler to provide custom processing of Memo fields instead of built-in Memo field editor.

Class

TCRDBGrid

Syntax

```
property OnMemoClick: TOnMemoClick;
```
Reserved.

5.4.1.2.5  OptionsEx Property

OptionsEx property provides control over TCRDBGrid-specific features. They complement inherited options found in Options property.

Class
TCRDBGrid

Syntax

```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 fill 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.
5.4.1.1.2.6 OptionsMenu Property

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.

Class
TCRDBGrid

Syntax

```delphi
property OptionsMenu: TPopupMenu;
```

5.4.1.1.3 Methods

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 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.</td>
</tr>
<tr>
<td>ActivateSearchEdit</td>
<td>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.</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</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</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.
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.

5.4.1.3.3 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.
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.

5.4.1.3.4 ApplyFilter Method

Call ApplyFilter method at run-time to update TCRDBGrid component taking into account filter statements defined for every column.

Class
**TCRDBGrid**

Syntax

```
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.

5.4.1.3.5 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

© 1997-2021 Devart. All Rights Reserved.

5.4.1.1.3.6 ClearFilters Method

Call CalcFilters method to clear all filter statements for grid columns.

Class
TCRDBGrid

Syntax

procedure ClearFilters;

© 1997-2021 Devart. All Rights Reserved.

5.4.1.1.3.7 ClearSorting Method

Call ClearSorting method to discard sorting previously applied to the grid data.

Class
TCRDBGrid

Syntax

procedure ClearSorting;

© 1997-2021 Devart. All Rights Reserved.

5.4.1.1.3.8 DataChanged Method

Call DataChanged method to update displayed data and status information for the grid.

Class
TCRDBGrid
5.4.1.3.9 GetGridSize Method

Call GetGridSize method to obtain width of a client area for the grid component.

Class
TCRDBGrid

Syntax

```delphi
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

```delphi
function GetTitleLevel(Level: integer): TRect;
```

Parameters

- **Level**

© 1997-2021 Devart. All Rights Reserved.
5.4.1.1.4 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

© 1997-2021 Devart. All Rights Reserved.

5.4.1.1.4.1 OnGetCellParams Event

Write an OnGetCellParams event handler to provide custom background colors while the grid redraws individual cells.

Class
- TCRDBGrid

Syntax

```pascal
property OnGetCellParams: TGetCellParamsEvent;
```

© 1997-2021 Devart. All Rights Reserved.

5.5 CRVio

This unit contains classes for HTTP connections.

Classes
## Classes

Classes in the `CRVio` unit.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>THttpOptions</code></td>
<td>This class is used to establish an HTTP connection.</td>
</tr>
<tr>
<td><code>TProxyOptions</code></td>
<td>This class is used to establish an HTTP connection through a proxy server.</td>
</tr>
</tbody>
</table>

## 5.5.1 Classes

### `THttpOptions` Class

This class is used to establish an HTTP connection.

For a list of all members of this type, see [THttpOptions](#) members.
Syntax

```
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>Enabled</td>
<td>Enables an HTTP connection.</td>
</tr>
<tr>
<td>Password</td>
<td>Holds the password for HTTP authorization.</td>
</tr>
<tr>
<td>ProxyOptions</td>
<td>Holds a TProxyOptions object that contains settings for a proxy connection.</td>
</tr>
<tr>
<td>TrustServerCertificate</td>
<td>Verifies the server certificate during an SSL handshake.</td>
</tr>
<tr>
<td>Url</td>
<td>Holds the URL of the PHP script for HTTP tunneling.</td>
</tr>
<tr>
<td>Username</td>
<td>Holds the username for HTTP authorization.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
5.5.1.1.2 Properties

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>Enabled</td>
<td>Enables an HTTP connection.</td>
</tr>
<tr>
<td>ProxyOptions</td>
<td>Holds a TProxyOptions 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>Password</td>
<td>Holds the password for HTTP authorization.</td>
</tr>
<tr>
<td>TrustServerCertificate</td>
<td>Verifies the server certificate during an SSL handshake.</td>
</tr>
<tr>
<td>Url</td>
<td>Holds the URL of the PHP script for HTTP tunneling.</td>
</tr>
<tr>
<td>Username</td>
<td>Holds the username for HTTP authorization.</td>
</tr>
</tbody>
</table>

See Also

- THttpOptions Class
- THttpOptions Class Members

Enables an HTTP connection.

Class

THttpOptions
5.5.1.1.2.2 Password Property

Holds the password for HTTP authorization.

Class

`THttpOptions`

Syntax

```php
property Password: string;
```

Remarks

The Password property holds the password for the password-protected directory that contains the HTTP tunneling script.

© 1997-2021

Devart. All Rights Reserved.

5.5.1.1.2.3 ProxyOptions Property

Holds a TProxyOptions object that contains settings for a proxy connection.

Class

`THttpOptions`

Syntax

```php
property ProxyOptions: TProxyOptions;
```

Remarks

The ProxyOptions property holds a TProxyOptions object that contains settings for a proxy connection.© 1997-2021

Devart. All Rights Reserved.
connection.

If it is necessary to connect to the server that resides in a different network, sometimes the client can only connect to it through a proxy server. In this case, besides the connection string, you have to set up ProxyOptions.

### 5.5.1.1.2.4 TrustServerCertificate Property

Verifies the server certificate during an SSL handshake.

**Class**

*THttpOptions*

**Syntax**

```delphi
property TrustServerCertificate: boolean default False;
```

**Remarks**

The TrustServerCertificate property specifies whether to verify the server certificate during an SSL handshake. When True, the MyDac bypasses walking the certificate chain to verify the certificate. The default value is False.

### 5.5.1.1.2.5 Url Property

Holds the URL of the PHP script for HTTP tunneling.

**Class**

*THttpOptions*

**Syntax**

```delphi
property Url: string;
```

**Remarks**

The Url property holds the URL of the PHP script for HTTP tunneling. For example, if the
script is located in the server root, the URL can be the following: http://server/tunnel.php.

### 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.

### 5.5.2 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.
5.5.1.2.1 Members

**TProxyOptions** class overview.

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>

See Also

- [TProxyOptions Class](#)
- [TProxyOptions Class Members](#)

© 1997-2021
Devart. All Rights Reserved.

Request Support   DAC Forum   Provide Feedback
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.

Class

TProxyOptions

Syntax

```property``` Username: string;

Remarks

The Username property holds the proxy username.

5.5.2 Enumerations

Enumerations in the CRVio unit.

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.

Request Support  DAC Forum  Provide Feedback
5.5.2.1 TIPVersion Enumeration

Specifies Internet Protocol version.

Unit

crvio

Syntax

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

• TMyConnectionOptions.IPVersion

5.6 DAALerter

This unit contains the base class for the TMyAlerter component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAALerter</td>
<td>A base class that defines functionality for database event notification.</td>
</tr>
</tbody>
</table>
5.6.1 Classes

Classes in the DAAlerter unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDAAlterer</td>
<td>A base class that defines functionality for database event notification.</td>
</tr>
</tbody>
</table>

5.6.1.1 TDAAlterer Class

A base class that defines functionality for database event notification.

For a list of all members of this type, see TDAAlterer members.

Unit
DAAlerter

Syntax

```
TDAAlterer = class(TComponent);
```

Remarks

TDAAlterer is a base class that defines functionality for descendant classes support database event notification. Applications never use TDAAlterer objects directly. Instead they use descendants of TDAAlterer.

The TDAAlterer component allows you to register interest in and handle events posted by a database server. Use TDAAlterer 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 the OnEvent handler is called.

© 1997-2021
Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
5.6.1.1.1 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>
</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 occurs in waiting process</td>
</tr>
</tbody>
</table>

5.6.1.1.2 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</td>
</tr>
</tbody>
</table>
5.6.1.2.1 Active Property

Used to determine if TDAlerter waits for messages.

Class
TDAlerter

Syntax

```plaintext
property Active: boolean default False;
```

Remarks

Check the Active property to know whether TDAlerter waits for messages or not. Set it to True to register events.

See Also

• Start
• Stop

5.6.1.2.2 AutoRegister Property

Used to automatically register events whenever connection opens.

Class
TDAlerter
Syntax

```java
property AutoRegister: boolean default False;
```

Remarks

Set the AutoRegister property to True to automatically register events whenever connection opens.

© 1997-2021 Devart. All Rights Reserved.

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>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>

See Also

- [TDAAlerter Class](#)
- [TDAAlerter Class Members](#)

© 1997-2021 Devart. All Rights Reserved.

5.6.1.1.3.1 SendEvent Method

Sends an event with Name and content Message.

Class

**TDAAlerter**

Syntax
procedure SendEvent(const EventName: string; const Message: string);

Parameters

EventName
  Holds the event name.

Message
  Holds the content Message of the event.

Remarks

Use SendEvent procedure to send an event with Name and content Message.

Start Method

Starts waiting process.

Class

TDAAlerter

Syntax

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

© 1997-2021
Devart. All Rights Reserved.
5.6.1.3.3 Stop Method

Stops waiting process.

Class
TDAAlerter

Syntax

procedure Stop;

Remarks
Call Stop method to end waiting process.

See Also
- Start

© 1997-2021
Devart. All Rights Reserved.

5.6.1.4 Events

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>OnError</td>
<td>Occurs if an exception occurs in waiting process</td>
</tr>
</tbody>
</table>

See Also
- TDAAlerter Class
- TDAAlerter Class Members

© 1997-2021
Devart. All Rights Reserved.
5.6.1.1.4.1 OnError Event

Occurs if an exception occurs in waiting process

**Class**

 TDAAlerter

**Syntax**

```none
property OnError: TAlerterErrorEvent;
```

**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.

5.7 DADump

This unit contains the base class for the TMyDump component.

**Classes**

<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>

**Types**

<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>
5.7.1 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>

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**

```delphi
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 descedants 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.

**TDADump** class overview.

### Properties

<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>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>Options</strong></td>
<td>Used to specify the behaviour of a TDADump component.</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>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Backup</strong></td>
<td>Dumps database objects to the TDADump.SQL property.</td>
</tr>
<tr>
<td><strong>BackupQuery</strong></td>
<td>Dumps the results of a particular query.</td>
</tr>
<tr>
<td><strong>BackupToFile</strong></td>
<td>Dumps database objects to the specified file.</td>
</tr>
<tr>
<td><strong>BackupToStream</strong></td>
<td>Dumps database objects to the stream.</td>
</tr>
<tr>
<td><strong>Restore</strong></td>
<td>Executes a script contained in the SQL property.</td>
</tr>
</tbody>
</table>
### RestoreFromFile
Executes a script from a file.

### RestoreFromStream
Executes a script received from the stream.

## Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnError</strong></td>
<td>Occurs when MySQL raises some error on <code>TDADump.Restore</code>.</td>
</tr>
<tr>
<td><strong>OnRestoreProgress</strong></td>
<td>Occurs to indicate the <code>TDADump.Restore</code>, <code>TDADump.RestoreFromFile</code>, or <code>TDADump.RestoreFromStream</code> method execution progress.</td>
</tr>
</tbody>
</table>

## Properties

Properties of the `TDADump` class.

For a complete list of the `TDADump` class members, see the `TDADump Members` topic.

## Public

### Name

<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>
</tbody>
</table>
Options

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>SQL</td>
<td>Used to set or get the dump script.</td>
</tr>
<tr>
<td>TableNames</td>
<td>Used to set the names of the tables to dump.</td>
</tr>
</tbody>
</table>

See Also

- **TDADump Class**
- **TDADump Class Members**

5.7.1.2.1 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 MyDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

**Note:** If TMySQLMonitor is used in the project and the TMySQLMonitor.Active property is set to False, the debug window is not displayed.

See Also
- TCustomDADataset.Debug
- TCustomDASQL.Debug
5.7.1.1.2.3 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 object names in generated SQL statements.</td>
</tr>
</tbody>
</table>

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

5.7.1.2.5 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

© 1997-2021
Devart. All Rights Reserved.

5.7.1.3 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

© 1997-2021 Devart. All Rights Reserved.

5.7.1.1.3.1 Backup Method

Dumps database objects to the SQL property.

Class

TDADump

Syntax

```
procedure Backup;
```

Remarks

Call the Backup method to dump database objects. The result script will be stored in the SQL property.
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

- **SQL**
- **Restore**
- **BackupToFile**
- **BackupToStream**
- **BackupQuery**
5.7.1.3.3 BackupToFile Method

Dumps database objects to the specified file.

Class

TDADump

Syntax

```delphi
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](#)

© 1997-2021 Devart. All Rights Reserved.

5.7.1.3.4 BackupToStream Method

Dumps database objects to the stream.

Class

TDADump

Syntax

```delphi
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

Class
TDADump

Syntax

```pascal
procedure Restore;
```

Remarks
Call the Restore method to execute a script contained in the SQL property.

See Also
- RestoreFromFile
- RestoreFromStream
- Backup
- SQL
5.7.1.3.6 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

5.7.1.3.7 RestoreFromStream Method

Executes a script received from the stream.

Class
TDADump

Syntax

```
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**

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>
<tbody>
<tr>
<td><strong>OnError</strong></td>
<td>Occurs when MySQL raises some error on <strong>TDADump.Restore</strong>.</td>
</tr>
<tr>
<td><strong>OnRestoreProgress</strong></td>
<td>Occurs to indicate the <strong>TDADump.Restore</strong>, <strong>TDADump.RestoreFromFile</strong>, or <strong>TDADump.RestoreFromStream</strong> method execution progress.</td>
</tr>
</tbody>
</table>

See Also
- [TDADump Class](#)
- [TDADump Class Members](#)
5.7.1.4.1 OnBackupProgress Event


Class

TDADump

Syntax

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

5.7.1.4.2 OnError Event

Occurs when MySQL raises some error on Restore.

Class

TDADump

© 2021 Devart

© 1997-2021

Devart. All Rights Reserved.
Syntax

```plaintext
property OnError: TOnErrorEvent;
```

Remarks

The OnError event occurs when MySQL 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.

### 5.7.1.4.3 OnRestoreProgress Event

Occurs to indicate the Restore, RestoreFromFile, or RestoreFromStream method execution progress.

Class

**TDADump**

Syntax

```plaintext
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**
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);

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>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>
For a complete list of the `TDADumpOptions` class members, see the `TDADumpOptions Members` topic.

**Published**

<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>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>

**See Also**
- `TDADumpOptions Class`
- `TDADumpOptions Class Members`

© 1997-2021 Devart. All Rights Reserved.

5.7.1.2.2.1 AddDrop Property

Used to add drop statements to a script before creating statements.

**Class**

`TDADumpOptions`

**Syntax**

```
property AddDrop: boolean default True;
```

**Remarks**

Use the AddDrop property to add drop statements to a script before creating statements.
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

```plaintext
property CompleteInsert: boolean default False;
```

Remarks
If the CompleteInsert property is set to True, SQL query will include the field names, for example:

```sql
INSERT INTO dept(deptno, dname, loc) VALUES ('10', 'ACCOUNTING', 'NEW YORK');
```
If False, it won't include the field names, for example:

```sql
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

```plaintext
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.
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.

5.7.2 Types

Types in the DADump unit.

Types

<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.
**TDABackupProgressEvent**

```object
procedure (Sender: TObject; ObjectName: string; ObjectNum: integer; ObjectCount: integer; Percent: integer)
```

**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.

---

**TDARestoreProgressEvent**

```object
procedure (Sender: TObject; Percent: integer)
```

**Parameters**

- **Sender**
  - An object that raised the event.

- **Percent**
  - The percentage of the whole restore script execution.

---

5.7.2.2 TDARestoreProgressEvent Procedure Reference

This type is used for the **TDADump.OnRestoreProgress** event.

---

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback  

© 2021 Devart
5.8 DALoader

This unit contains the base class for the TMyLoader 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>
<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>

5.8.1 Classes

Classes in the DALoader unit.

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>
</tbody>
</table>
### TDALoader

This class allows loading external data into database.

### TDALoaderOptions

Allows loading external data into database.

---

## 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

```delphi
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>
</table>

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback
5.8.1.1.2 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>FieldType</td>
<td>Used to specify the types of values that will be loaded.</td>
</tr>
<tr>
<td>Name</td>
<td>Used to specify the field name of loading table.</td>
</tr>
</tbody>
</table>

See Also
- TDAColumn Class
- TDAColumn Class Members

5.8.1.2.1 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
• FieldType

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

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
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. All Rights Reserved.

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.

See Also
- `TMyLoader`

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>Used to add a <code>TDAColumn</code> object for each field that will be loaded.</td>
</tr>
<tr>
<td>Connection</td>
<td>property. Used to specify TCustomDAConection 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 TDALoader.LoadFromDataSet Set method is needed.</td>
</tr>
<tr>
<td>OnPutData</td>
<td>Occurs when putting loading data by rows is needed.</td>
</tr>
</tbody>
</table>

Properties 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>Columns</td>
<td>Used to add a <strong>TDAColumn</strong> object for each field that will be loaded.</td>
</tr>
<tr>
<td>Connection</td>
<td>property. Used to specify TCustomDACConnection 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>

See Also
- **TDALoader Class**
- **TDALoader Class Members**

© 1997-2021 Devart. All Rights Reserved.  
[Request Support]  [DAC Forum]  [Provide Feedback]
5.8.1.3.2.1 Columns Property

Used to add a TDAColumn object for each field that will be loaded.

Class

TDALoader

Syntax

| 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

5.8.1.3.2.2 Connection Property

property. Used to specify TCustomDAConnection 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
5.8.1.3.3  TableName Property

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
- TCustomDACConnection.GetTableNames

© 1997-2021 Devart. All Rights Reserved.

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
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

```
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`

LoadFromDataSet Method

Loads data from the specified dataset.

Class

`TDALoader`

Syntax

```pascal
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 `OnPutData` and `OnGetColumnData` before calling this method.

PutColumnData Method

Puts the value of individual columns.

Class

`TDALoader`
Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PutColumnData(Col: integer; Row: integer; const Value: variant)</td>
<td>Puts the value of individual columns by the column index.</td>
</tr>
<tr>
<td>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>

Puts the value of individual columns by the column index.

Class

TDALoader

Syntax

```procedure PutColumnData(Col: integer; Row: integer; const Value: variant); overload; virtual;```

Parameters

- **Col**
  - Holds the index of a loading column. The first column has index 0.
- **Row**
  - Holds the number of loading row. Row starts from 1.
- **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
Puts the value of individual columns by the column name.

Class

**TDALoader**

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>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.LoadFromData</code> Set method is needed.</td>
</tr>
<tr>
<td>OnPutData</td>
<td>Occurs when putting loading data by rows is needed.</td>
</tr>
</tbody>
</table>
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.

```
procedure TFormMain.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;
  end;
end;
```
else
  Value := Null;
end;
end
else
  EOF := True;
end;

See Also
- OnPutData
- Load

5.8.1.3.4.2  OnProgress Event

Occurs if handling data loading progress of the LoadFromDataSet method is needed.

Class
TDALoader

Syntax

property OnProgress: TLoaderProgressEvent;

Remarks
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

```plaintext
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. TMyLoader will flush data to MySQL when it is needed (see TMyLoader.RowsPerQuery).

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;
```

See Also

- TDALoader.PutColumnData
- Load
- OnGetColumnData

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

**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 MyDAC to fill the buffer with null values after loading a row to the database.</td>
</tr>
</tbody>
</table>

### 5.8.1.4.2 Properties

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>UseBlankValues</td>
<td>Forces MyDAC 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**
5.8.1.4.2.1 UseBlankValues Property

Forces MyDAC to fill the buffer with null values after loading a row to the database.

Class

TDALoaderOptions

Syntax

property UseBlankValues: boolean default True;

Remarks

Used to force MyDAC 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>

© 1997-2021
Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
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

```
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**

```pascal
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 TMyScript component.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TDAScript</strong></td>
<td>Makes it possible to execute several SQL statements one by one.</td>
</tr>
<tr>
<td><strong>TDAStatement</strong></td>
<td>This class has attributes and methods for controlling single SQL statement of a script.</td>
</tr>
<tr>
<td><strong>TDAStatements</strong></td>
<td>Holds a collection of <strong>TDAStatement</strong> 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.

<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>
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 descendant 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.

See Also

• TCustomDASQL

© 1997-2021
Devart. All Rights Reserved.

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</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BreakExec</td>
<td>Stops script execution.</td>
</tr>
<tr>
<td>ErrorOffset</td>
<td>Used to get the offset of the statement if the Execute method raised an exception.</td>
</tr>
<tr>
<td>Execute</td>
<td>Executes a script.</td>
</tr>
<tr>
<td>ExecuteFile</td>
<td>Executes SQL statements contained in a file.</td>
</tr>
</tbody>
</table>
**ExecuteNext**
Executes the next statement in the script and then stops.

**ExecuteStream**
Executes SQL statements contained in a stream object.

**FindMacro**
Finds a macro with the specified name.

**MacroByName**
Finds a macro with the specified name.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterExecute</td>
<td>Occurs after a SQL script execution.</td>
</tr>
<tr>
<td>BeforeExecute</td>
<td>Occurs when taking a specific action before executing the current SQL statement is needed.</td>
</tr>
<tr>
<td>OnError</td>
<td>Occurs when MySQL raises an error.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  

**5.9.1.2 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>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</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 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>

### See Also
- [TDAScript Class](#)
- [TDAScript Class Members](#)
5.9.1.1.2.1 Connection Property

Used to specify the connection in which the script will be executed.

Class

TDAScript

Syntax

```
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 `TCustomDACConnection` objects.

At run-time, set the Connection property to reference an existing `TCustomDACConnection` object.

See Also

- `TCustomDACConnection`

© 1997-2021 Devart. All Rights Reserved.

5.9.1.1.2.2 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
5.9.1.2.3  Debug Property

Used to display the script execution and all its parameter values.

Class
TDAScript

Syntax

```plaintext
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 MyDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

**Note:** If TMySQLMonitor is used in the project and the TMySQLMonitor.Active property is set to False, the debug window is not displayed.

5.9.1.2.4  Delimiter Property

Used to set the delimiter string that separates script statements.

Class
TDAScript

Syntax

```plaintext
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.

5.9.1.1.2.5 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.
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**

© 1997-2021 Devart
5.9.1.1.2.9  SQL Property

Used to get or set script text.

Class
TDAScript

Syntax

```delphi
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

```delphi
property StartLine: Int64;
```

Remarks
Use the StartLine property to get the current statement start line number in a script.

5.9.1.1.2.11  StartOffset Property

Used to get the offset in the first line of the current statement.

Class
**TDAScript**

**Syntax**

```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**

```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.

© 1997-2021 Devart. All Rights Reserved.

5.9.1.2.13 Statements Property

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:

```
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:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
var
  i: integer;
begin
  with Script do
  begin
    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

© 1997-2021 Devart. All Rights Reserved.
## Name

<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>

### See Also

- TDAScript Class
- TDAScript Class Members

© 1997-2021 Devart. All Rights Reserved.

### 5.9.1.1.3.1 BreakExec Method

Stops script execution.

**Class**

TDAScript

**Syntax**

```
procedure BreakExec; virtual;
```

**Remarks**

Call the BreakExec method to stop script execution.

© 1997-2021 Devart. All Rights Reserved.
5.9.1.3.2 ErrorOffset Method

Used to get the offset of the statement if the Execute method raised an exception.

Class
TDAScript

Syntax

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

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 MySQL raises an error, the OnError event occurs.

See Also
• ExecuteNext
5.9.1.3.4 ExecuteFile Method

Executes SQL statements contained in a file.

Class
TDAScript

Syntax

```pascal
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

```pascal
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 MySQL raises an error, the OnError event occurs.

See Also

- Execute
- OnError
- ErrorOffset

Syntax

```delphi
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.
Syntax

```plaintext
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`

5.9.1.1.3.8 MacroByName Method

Finds a macro with the specified name.

**Class**

`TDAScript`

**Syntax**

```plaintext
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**

© 1997-2021 Devart. All Rights Reserved.

© 2021 Devart
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`

5.9.1.1.4 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>AfterExecute</td>
<td>Occurs after a SQL script execution.</td>
</tr>
<tr>
<td>BeforeExecute</td>
<td>Occurs when taking a specific action before executing the current SQL statement is needed.</td>
</tr>
<tr>
<td>OnError</td>
<td>Occurs when MySQL raises an error.</td>
</tr>
</tbody>
</table>

See Also
- `TDAScript Class`
- `TDAScript Class Members`
5.9.1.4.1 AfterExecute Event

Occurs after a SQL script execution.

Class

TDAScript

Syntax

property AfterExecute: TAfterStatementExecuteEvent;

Remarks

Occurs after a SQL script has been executed.

See Also

• Execute

© 1997-2021
Devart. All Rights Reserved.

5.9.1.4.2 BeforeExecute Event

Occurs when taking a specific action before executing the current SQL statement is needed.

Class

TDAScript

Syntax

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.
5.9.1.4.3 OnError Event

Occurs when MySQL raises an error.

Class

TDAScript

Syntax

property OnError: TOnErrorEvent;

Remarks

Occurs when MySQL 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

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
5.9.1.2.1 Members

**TDAScript**

**TDASTatements**

© 1997-2021 Devart. All Rights Reserved.

**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 SQL statement.</td>
</tr>
<tr>
<td>Script</td>
<td>Used to determine the TDAScript object the SQL Statement belongs to.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to get or set the text of an SQL statement.</td>
</tr>
<tr>
<td>StartLine</td>
<td>Used to determine the number of the first statement line in a script.</td>
</tr>
<tr>
<td>StartOffset</td>
<td>Used to get the offset in the first line of a statement.</td>
</tr>
<tr>
<td>startPos</td>
<td>Used to get the start position of the statement in a script.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Executes a statement.</td>
</tr>
</tbody>
</table>
Properties of the **TDASStatement** class.

For a complete list of the **TDASStatement** class members, see the [TDASStatement 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 SQL statement.</td>
</tr>
<tr>
<td>Script</td>
<td>Used to determine the TDAScript object the SQL Statement belongs to.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to get or set the text of an SQL statement.</td>
</tr>
<tr>
<td>StartLine</td>
<td>Used to determine the number of the first statement line in a script.</td>
</tr>
<tr>
<td>StartOffset</td>
<td>Used to get the offset in the first line of a statement.</td>
</tr>
<tr>
<td>StartPos</td>
<td>Used to get the start position of the statement in a script.</td>
</tr>
</tbody>
</table>

See Also

- [TDASStatement Class](#)
- [TDASStatement Class Members](#)
5.9.1.2.2.1 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.

5.9.1.2.2.2 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.

5.9.1.2.2.3 EndPos Property

Used to get the end position of the statement in a script.

Class

TDAStatement

© 2021 Devart
**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.

5.9.1.2.2.4  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

Class
TDAScript

Syntax

```property script: TDAScript;```

Remarks
Use the Script property to determine the TDAScript object the SQL Statement belongs to.

Class
TDAScript

Syntax

```property SQL: string;```

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

TDASTatement

Syntax

```pascal
property StartLine: integer;
```

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

TDASTatement

Syntax

```pascal
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
**TDASatement**

**Syntax**

```property```
property StartPos: integer;
```property```

**Remarks**

Use the StartPos property to get the start position of the statement (the position of the first statement character) in a script.

© 1997-2021 Devart. All Rights Reserved.

5.9.1.2.3 Methods

Methods of the **TDASatement** class.

For a complete list of the **TDASatement** class members, see the **TDASatement Members** topic.

**Public**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Executes a statement.</td>
</tr>
</tbody>
</table>

See Also

- **TDASatement Class**
- **TDASatement Class Members**

© 1997-2021 Devart. All Rights Reserved.

5.9.1.2.3.1 Execute Method

Executes a statement.

**Class**

**TDASatement**

**Syntax**

```procedure```
procedure Execute;
```procedure```
Remarks
Use the Execute method to execute a statement.

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

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**
- **TDAStatements Class**
- **TDAStatements Class Members**

5.9.1.3.2.1 Items Property (Indexer)

Used to access separate script statements.

**Class**

**TDAStatements**

**Syntax**

```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**.
## 5.9.2 Types

Types in the **DAScript** unit.

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TAfterStatementExecuteEvent</strong></td>
<td>This type is used for the <code>TDAScript.AfterExecute</code> event.</td>
</tr>
<tr>
<td><strong>TBeforeStatementExecuteEvent</strong></td>
<td>This type is used for the <code>TDAScript.BeforeExecute</code> event.</td>
</tr>
<tr>
<td><strong>OnErrorEvent</strong></td>
<td>This type is used for the <code>TDAScript.OnError</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

```delphi
TAfterStatementExecuteEvent = procedure (Sender: TObject; SQL: string) 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**

```plaintext
TBeforeStatementExecuteEvent = procedure (Sender: TObject; var SQL: string; 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**

```plaintext
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.
5.9.3 Enumerations

Enumerations in the DAScript unit.

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>

5.9.3.1 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 TMySQLMonitor 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 displayed.</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</td>
</tr>
<tr>
<td></td>
<td>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.
### 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 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>OnSQL</td>
<td>Occurs when tracing of SQL activity on database components is needed.</td>
</tr>
</tbody>
</table>

For a complete list of the **TCustomDASQLMonitor** class members, see the [TCustomDASQLMonitor Members](#) topic.
### Options

Used to include the desired properties for TCustomDASQLMonitor.

### TraceFlags

Used to specify which database operations the monitor should track in an application at runtime.

---

**See Also**
- TCustomDASQLMonitor Class
- TCustomDASQLMonitor Class Members

---

#### 5.10.1.1.2.1 Active Property

Used to activate monitoring of SQL.

**Class**

TCustomDASQLMonitor

**Syntax**

```plaintext
property Active: boolean default True;
```

**Remarks**

Set the Active property to True to activate monitoring of SQL.

---

**See Also**
- OnSQL

---

#### 5.10.1.1.2.2 DBMonitorOptions Property

Used to set options for dbMonitor.

**Class**

TCustomDASQLMonitor
5.10.1.1.2.3 Options Property

Used to include the desired properties for TCustomDASQLMonitor.

Class

TCustomDASQLMonitor

Syntax

**property** DBMonitorOptions: TDBMonitorOptions;

Remarks

Use DBMonitorOptions to set options for dbMonitor.

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
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.

5.10.1.1.3.1 OnSQL Event

Occurs when tracing of SQL activity on database components is needed.

Class

TCustomDASQLMonitor
Syntax

```delphi
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`

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

```delphi
TDBMonitorOptions = class(TPersistent);
```

5.10.1.2.1  Members

`TDBMonitorOptions` class overview.

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>
</tbody>
</table>
### 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.

© 1997-2021 Devart. All Rights Reserved.

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

```delphi
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.

© 1997-2021 Devart. All Rights Reserved.

5.10.1.2.2.4 SendTimeout Property

Used to set timeout for sending events to dbMonitor.

Class

**TDBMonitorOptions**

Syntax

```delphi
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.

© 1997-2021 Devart. All Rights Reserved.
### 5.10.2 Types

Types in the **DASQLMonitor** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATraceFlags</td>
<td>Represents the set of <strong>TDATraceFlag</strong>.</td>
</tr>
<tr>
<td>TMonitorOptions</td>
<td>Represents the set of <strong>TMonitorOption</strong>.</td>
</tr>
<tr>
<td>TOnSQLEvent</td>
<td>This type is used for the <strong>TCustomDASQLMonitor.On SQL</strong> event.</td>
</tr>
</tbody>
</table>

#### 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**

```pascal
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.

© 1997-2021 Devart. All Rights Reserved.

5.10.3 **Enumerations**

Enumerations in the `DASQLMonitor` unit.

**Enumerations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TDATraceFlag</code></td>
<td>Use TraceFlags to specify which database operations the monitor should track in an application at runtime.</td>
</tr>
<tr>
<td><code>TMonitorOption</code></td>
<td>Used to define where information from SQLMonitor will be displayed.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
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

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>If this flag is set, then just before sending a query to the server, OnSQL event is called additionally. The difference from usual call is that the query is already completely decoded, i.e. parameters are quoted and included into the text of the query. If to use MySQL 4.1 protocol with preparing, a value of this flag will be ignored.</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>tfPrepare</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. Request Support  DAC Forum  Provide Feedback
5.10.3.2 TMonitorOption Enumeration

Used to define where information from SQLMonitor will be displayed.

Unit

DASQLMonitor

Syntax

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 properties in SQL-related components to True to let TCustomDASQLMonitor 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>
<tr>
<td>moDialog</td>
<td>Debug information is displayed in debug window.</td>
</tr>
<tr>
<td>moHandled</td>
<td>Component handle is included into the event description string.</td>
</tr>
<tr>
<td>moSQLMonitor</td>
<td>Debug information is displayed in Borland SQL Monitor.</td>
</tr>
</tbody>
</table>

5.11 DBAccess

This unit contains base classes for most of the components.

Classes

<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>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</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 TDAConditions list.</td>
</tr>
<tr>
<td>TDAConditions</td>
<td>Holds a collection of TDACondition objects.</td>
</tr>
<tr>
<td>TDAConnnectionOptions</td>
<td>This class allows setting up the behaviour of the TDAConnection class.</td>
</tr>
<tr>
<td>TDAConnectionSSLOptions</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 metainformation of the specified database objects in the form of dataset.</td>
</tr>
<tr>
<td>TDAParam</td>
<td>A class that forms objects to</td>
</tr>
</tbody>
</table>
represent the values of the parameters set.

**TDAParams**

This class is used to manage a list of TDAParam objects for an object that uses field parameters.

**TDATransaction**

A base class that implements functionality for controlling transactions.

**TMacro**

Object that represents the value of a macro.

**TMacros**

Controls a list of TMacro objects for the `TCustomDASQL.Macros` or `TCustomDADataset` components.

**TPoolingOptions**

This class allows setting up the behaviour of the connection pool.

**TSmartFetchOptions**

Smart fetch options are used to set up the behavior of the SmartFetch mode.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TAfterExecuteEvent</code></td>
<td>This type is used for the <code>TCustomDADataset.AfterExecute</code> and <code>TCustomDASQL.AfterExecute</code> events.</td>
</tr>
<tr>
<td><code>TAfterFetchEvent</code></td>
<td>This type is used for the <code>TCustomDADataset.AfterFetch</code> event.</td>
</tr>
<tr>
<td><code>TBeforeFetchEvent</code></td>
<td>This type is used for the <code>TCustomDADataset.BeforeFetch</code> event.</td>
</tr>
<tr>
<td><code>TConnectionLostEvent</code></td>
<td>This type is used for the <code>TCustomDACConnection.OnConnectionLost</code> event.</td>
</tr>
<tr>
<td><code>TDAConnectionErrorEvent</code></td>
<td>This type is used for the <code>TCustomDACConnection.OnError</code> event.</td>
</tr>
</tbody>
</table>
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.

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>

Variables

<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 AddWhere, DeleteWhere, and SetOrderBy, all subsequent changes of the SQL property will not be reflected in the BaseSQL 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 TCustomDADataSet.BaseSQL property is used to complete the refresh SQL statement, if the manually assigned</td>
</tr>
</tbody>
</table>
## Classes

Classes in the **DBAccess** unit.

<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 components and data-aware controls on a form.</td>
</tr>
<tr>
<td><strong>TCustomConnectDialog</strong></td>
<td>A base class for the connect dialog components.</td>
</tr>
<tr>
<td><strong>TCustomDAConnection</strong></td>
<td>A base class for components used to establish connections.</td>
</tr>
<tr>
<td><strong>TCustomDADataset</strong></td>
<td>Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.</td>
</tr>
<tr>
<td><strong>TCustomDASQL</strong></td>
<td>A base class for components executing SQL statements that do not return result sets.</td>
</tr>
<tr>
<td><strong>TCustomDAUpdateSQL</strong></td>
<td>A base class for components that provide DML statements for more flexible control over data modifications.</td>
</tr>
<tr>
<td><strong>TDACCondition</strong></td>
<td>Represents a condition from the <strong>TDACConditions</strong> list.</td>
</tr>
<tr>
<td><strong>TDACConditions</strong></td>
<td>Holds a collection of <strong>TDACCondition</strong> objects.</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TDAConnectionOptions</td>
<td>This class allows setting up the behaviour of the TDAConnection class.</td>
</tr>
<tr>
<td>TDAConnectionSSLOptions</td>
<td>This class is used to set up the SSL options.</td>
</tr>
<tr>
<td>TDADebugOptions</td>
<td>This class allows setting up the behaviour of the TDADebugOptions 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 metainformation of the specified database objects in the form of dataset.</td>
</tr>
<tr>
<td>TDAParam</td>
<td>A class that forms objects to represent the values of the parameters set.</td>
</tr>
<tr>
<td>TDAParams</td>
<td>This class is used to manage a list of TDAParam objects for an object that uses field parameters.</td>
</tr>
<tr>
<td>TDAParams</td>
<td>A base class that implements functionality for controlling transactions.</td>
</tr>
<tr>
<td>TMacro</td>
<td>Object that represents the value of a macro.</td>
</tr>
<tr>
<td>TMacros</td>
<td>Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDADataset components.</td>
</tr>
<tr>
<td>TPoolingOptions</td>
<td>This class allows setting up the behaviour of the connection pool.</td>
</tr>
<tr>
<td>TSmartFetchOptions</td>
<td>Smart fetch options are used to set up the behavior</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>

© 1997-2021 Devart. All Rights Reserved.
5.11.1.1.2 Properties

Properties of the EDAError class.

For a complete list of the EDAError class members, see the EDAError Members topic.

Public

<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

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 MySQL. This value is always positive.


See Also

• EMyError

5.11.1.2 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.

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.

5.11.1.2.1 Members

**TCRDataSource** class overview.

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**

```pascal
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.

5.11.1.3.1 Members

**TCustomConnectDialog** class overview.

**Properties**

<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>Execute</td>
<td>Displays the connect dialog and calls the connection's Connect method when the Connect button is clicked.</td>
</tr>
<tr>
<td>GetServerList</td>
<td>Retrieves a list of available server names.</td>
</tr>
</tbody>
</table>
5.11.1.3.2 Properties

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>
<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>

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

```plaintext
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**

```property``` ConnectButton: ```string```

**Remarks**

Use the ConnectButton property to specify the label for the Connect button.

**Class**

**TCustomConnectDialog**

**Syntax**

```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 MyDAC 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

```plaintext
property LabelSet: TLabelSet default lsEnglish;
```

Remarks

Use the LabelSet property to set the language of labels and buttons captions.
The default value is lsEnglish.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.3.2.6 PasswordLabel Property

Used to specify a prompt for password edit.

Class

TCustomConnectDialog

Syntax

```plaintext
property PasswordLabel: string;
```

Remarks

Use the PasswordLabel property to specify a prompt for password edit.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.3.2.7 Retries Property

Used to indicate the number of retries of failed connections.

Class

TCustomConnectDialog
5.11.1.3.2.8 SavePassword Property

Used for the password to be displayed in ConnectDialog in asterisks.

Class

TCustomConnectDialog

Syntax

```
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.
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.

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.
5.11.1.3.3 Methods

Methods 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>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.
Request Support DAC Forum Provide Feedback

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

```pascal
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 \texttt{Retries} times.

5.11.1.3.2 \texttt{GetServerList} Method

Retrieves a list of available server names.

Class

\texttt{TCustomConnectDialog}

Syntax

\begin{verbatim}
procedure GetServerList(List: TStrings); virtual;
\end{verbatim}

Parameters

\textit{List}

Holds a list of available server names.

Remarks

Call the \texttt{GetServerList} method to retrieve a list of available server names. It is particularly relevant for writing custom login form.

See Also

\begin{itemize}
  \item \texttt{DialogClass}
\end{itemize}

5.11.1.4 \texttt{TCustomDACConnection} Class

A base class for components used to establish connections.

For a list of all members of this type, see \texttt{TCustomDACConnection} members.

Unit

\texttt{DBAccess}

Syntax

\begin{verbatim}
TCustomDACConnection = class(TCustomConnection);
\end{verbatim}
Remarks

TCustomDACConnection is a base class for components that establish connection with database, provide customised login support, and perform transaction control.

Do not create instances of TCustomDACConnection. To add a component that represents a connection to a source of data, use descendants of the TCustomDACConnection class.

5.11.1.4.1 Members

**TCustomDACConnection** class overview.

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, 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>
</tbody>
</table>
## Username

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>Used to supply a user name for login.</td>
</tr>
</tbody>
</table>

## Methods

<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>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 <strong>TCustomDASQLMonitor</strong> 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>
</tbody>
</table>
**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>

---

5.11.1.4.2 Properties

Properties 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>ConnectDialog</td>
<td>Allows to link a <strong>TCustomConnectDialog</strong> 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</td>
</tr>
</tbody>
</table>
### ConnectDialog Property

 Allows to link a **TCustomConnectDialog** component.

#### Class

**TCustomDAConnection**

#### Syntax

```pascal
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

MyDAC 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 TMyConnection component.</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Charset</td>
<td>Used to set the character set that MyDAC uses to read and write character data.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Used to specify the Network protocol of connection with MySQL server.</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>Compress</td>
<td>Used to apply compression on transferring data.</td>
</tr>
<tr>
<td>Embedded</td>
<td>Used to specify the server that will be used to connect.</td>
</tr>
<tr>
<td>Direct</td>
<td>Used to work without using MySQL client library (libmysql.dll).</td>
</tr>
<tr>
<td>IPVersion</td>
<td>Used to specify the version of the Internet Protocol.</td>
</tr>
<tr>
<td>P:Devart.Mydac.TMyConnectionSSLOptions.CACert</td>
<td>Holds the pathname to the certificate authority file.</td>
</tr>
<tr>
<td>P:Devart.Mydac.TMyConnectionSSLOptions.CipherList</td>
<td>Holds the list of allowed ciphers to use for SSL encryption.</td>
</tr>
<tr>
<td>P:Devart.Mydac.TMyConnectionSSLOptions.Key</td>
<td>Holds the pathname to the key file.</td>
</tr>
<tr>
<td>Username</td>
<td>Holds the user name for HTTP authorization.</td>
</tr>
<tr>
<td>Password</td>
<td>Holds the password for HTTP authorization.</td>
</tr>
<tr>
<td>URL</td>
<td>Holds the url of the tunneling PHP script.</td>
</tr>
<tr>
<td>Hostname</td>
<td>Holds the host name or IP address to connect to proxy server.</td>
</tr>
<tr>
<td>Username</td>
<td>Holds the proxy server account name.</td>
</tr>
<tr>
<td>Password</td>
<td>Holds the password for the proxy server account.</td>
</tr>
<tr>
<td>Port</td>
<td>Used to specify the port number for TCP/IP connection with proxy server.</td>
</tr>
</tbody>
</table>

See Also
- Password
- Username
5.11.1.4.2.3  **ConvertEOL Property**

Allows customizing line breaks in string fields and parameters.

**Class**

**TCustomDAConnection**

**Syntax**

```plaintext
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.

5.11.1.4.2.4  **InTransaction Property**

Indicates whether the transaction is active.

**Class**

**TCustomDAConnection**

**Syntax**

```plaintext
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

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 MyDacVcl unit appears to the uses clause.

Options Property

Specifies the connection behavior.

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 <code>TMemDataSet.IndexFieldNames</code> 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 <code>OnConnectionLost</code> 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](#)

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**

```plaintext
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.

5.11.1.4.2.9 PoolingOptions Property

Specifies the behaviour of connection pool.

Class

TCustomDAConnection

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 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...</td>
</tr>
</tbody>
</table>

© 2021 Devart
### Server Property

Serves to supply the server name for login.

**Class**

`TCustomDAConnection`

**Syntax**

```
property Server: string;
```

**Remarks**

Use the Server property to supply server name to handle server’s request for a login. If this property is not set, MyDAC tries to connect to Localhost.

The Server property can be used only if `TMyConnectionOptions.Embedded` is set to False.

**See Also**

- `Username`
- `Password`
- `TMyConnection.Port`

### Username Property

Used to supply a user name for login.

**Class**
**TCustomDAConnection**

**Syntax**

```
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, MyDAC tries to connect with the empty 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>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>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</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](#)

© 1997-2021
Devarl. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.11.1.4.3.1 ApplyUpdates Method

Applies changes in datasets.

Class
**TCustomDAConnection**

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Applies changes from all active datasets.</td>
</tr>
<tr>
<td><strong>ApplyUpdates(const DataSets: array of TCustomDADataset)</strong></td>
<td>Applies changes from the specified datasets.</td>
</tr>
</tbody>
</table>

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`

© 1997-2021
Devart. All Rights Reserved.
TCustomDAConnection

**Syntax**

```idl
procedure ApplyUpdates(const DataSets: array of TCustomDAQDataSet); 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.

**Commit Method**

Commits current transaction.

**Class**

TCustomDAConnection

**Syntax**

```idl
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**
5.11.1.4.3.3 Connect Method

Establishes a connection to the server.

Class

`TCustomDAConnection`

Syntax

```plaintext
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

```pascal
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

```pascal
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 rolls back 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.
5.11.1.4.3.6 ExecProc Method

Allows to execute stored procedure or function providing its name and parameters.

Class

TCustomDAConnection

Syntax

```
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):

Example 1)
CREATE procedure MY_SUM (      A INTEGER,      B INTEGER)RETURNS (      RESULT INTEGER)
as
begin
    Result = a + b;
end;
Example 2)
Label2.Caption:= MyMyConnection1.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

function ExecProcEx(const Name: string; const Params: array of variant): variant; virtual;
**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:

```
Connection.ExecProcEx('Some_Stored_Procedure', ['Param_Name1', 'Param_Value1', 'Param_Name4', 'Param_Value4']);
```

**See Also**

- ExecSQL
- ExecSQLEx
5.11.4.3.8 ExecSQL Method

Executes a SQL statement with parameters.

Class
TCustomDACConnection

Syntax

```delphi
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.

**Note:** If a query doesn't have parameters (Params.Count = 0), this query will be executed faster.

See Also

- ExecSQLEX
- ExecProc
- TCustomMyConnection.ExecSQL
5.11.1.4.3.9  ExecSQLEx Method

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
MyConnection.ExecSQLEx('begin :A:= :B + :C; end;', ['A', 0, 'B', 5, 'C', 3]);
A:= MyConnection.ParamByName('A').AsInteger;

See Also
- ExecSQL

Class
TCustomDAConnection

Syntax

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
- TCustomMyConnection.Database
5.11.1.4.3.11 GetKeyFieldNames Method

Provides a list of available key field names.

Class

TCustomDAConnection

Syntax

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

© 1997-2021 Devart. All Rights Reserved.

5.11.1.4.3.12 GetStoredProcNames Method

Returns a list of stored procedures from the server.

Class

TCustomDAConnection

Syntax

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](#)

---

**5.11.1.4.3.13 GetTableNames Method**

Provides a list of available tables names.

**Class**

TCustomDACConnection

**Syntax**

```delphi
procedure GetTableNames(List: TStrings; AllTables: boolean = False; OnlyTables: boolean = False); virtual;
```

**Parameters**

*List*

A TStrings descendant that will be filled with table names.

*OnlyTables*

A TStrings descendant that will be filled with table names.
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](#)

Sends a specified message through the TCustomDASQLMonitor component.

**Class**

TCustomDAConnection

**Syntax**

```plaintext
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.
### Ping Method

Used to check state of connection to the server.

**Class**

`TCustomDAConnection`

**Syntax**

```pascal
procedure ping;
```

**Remarks**

The method is used for checking server connection state. Use the Ping method to prevent server from closing connection on exceeding the wait_timeout value of the MySQL server. For this, call the Ping method in intervals smaller than the server wait_timeout value.

---

### RemoveFromPool Method

Marks the connection that should not be returned to the pool after disconnect.

**Class**

`TCustomDAConnection`

**Syntax**

```pascal
procedure RemoveFromPool;
```

**Remarks**

Call the RemoveFromPool method to mark the connection that should be deleted after disconnect instead of returning to the connection pool.
5.11.1.4.3.17 Rollback Method

Discards all current data changes and ends transaction.

Class
TCustomDAConnection

Syntax

```
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
- TCustomMyDataSet.FetchAll

© 1997-2021 Devart. All Rights Reserved.

5.11.1.4.3.18 StartTransaction Method

Begins a new user transaction.

Class
TCustomDAConnection

Syntax
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`
- `TCustomMyConnection.IsolationLevel`

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`
5.11.1.4.4.1  OnConnectionLost Event

This event occurs when connection was lost.

Class
TCustomDAConnection

Syntax

```delphi
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.

5.11.1.4.4.2  OnError Event

This event occurs when an error has arisen in the connection.

Class
TCustomDAConnection

Syntax

```delphi
property OnError: TDACConnectionErrorEvent;
```

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.
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

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

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL</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></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 mapping rules</td>
</tr>
<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>DetailFields</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></td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>FetchRows</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></td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</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 fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>IsQuery</strong></td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeyFields</strong></td>
<td>Used to build SQL</td>
</tr>
</tbody>
</table>
statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 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 TCustomDADataSet object.</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>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td>Prepared</td>
<td>(inherited from TMemDataSet) Determines whether a query is prepared for execution or</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Ranged</strong></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><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 TCustomDACDataSet.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</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddWhere</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 updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong></td>
<td>Breaks execution of the SQL statement on the server.</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>CreateBlobStream</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>DeferredPost</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteWhere</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 record.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</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>Execute</strong></td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td><strong>Fetched</strong></td>
<td>Used to find out whether TCustomDADDataSet has fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong></td>
<td>Used to learn whether TCustomDADDataSet is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong></td>
<td>Used to learn whether TCustomDADDataSet is fetching all rows to the end.</td>
</tr>
<tr>
<td><strong>FindKey</strong></td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindNearest</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 KeyValues parameter.</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>Determines if a parameter with the specified name exists in a dataset.</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></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</td>
</tr>
<tr>
<td>Method</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>
</tbody>
</table>
SaveToXML (inherited from TMemDataSet)  Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

SetOrderBy  Builds an ORDER BY clause of a SELECT statement.

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  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>AfterExecute</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td>Event Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</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>(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>

5.11.1.5.2 Properties

Properties 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>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 TMemDataSet) 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><strong>DataTypeMap</strong></td>
<td>Used to set data type mapping rules</td>
</tr>
<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>DetailFields</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></td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>FetchRows</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></td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</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 TMemDataSet)</td>
<td>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>IsQuery</strong></td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from TMemDataSet)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeyFields</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>LocalConstraints</strong> (inherited from TMemDataSet)</td>
<td>Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of...</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 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><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>
See Also

- TCustomDADataSet Class
- TCustomDADataSet Class Members

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

5.11.1.5.2.2 Conditions Property

Used to add WHERE conditions to a query

Class
**TCustomDADataSet**

**Syntax**

```plaintext
property Conditions: TDAConditions stored False;
```

**Remarks**

Use the `Conditions` property to specify a connection object that will be used to connect to a data store.

**See Also**

- [TDAConditions](#)

© 1997-2021 Devart. All Rights Reserved.

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**

```plaintext
property Connection: TCustomDACollection;
```

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.2.4 Data Type Map Property

Used to set data type mapping rules

**Class**

**TCustomDADataSet**

© 2021 Devart
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

```
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 MyDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

**Note:** If TMySQLMonitor is used in the project and the TMySQLMonitor.Active property is set to False, the debug window is not displayed.

See Also

- TCustomDASQL.Debug
Class

TCustomDADataSet

Syntax

```plaintext
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

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.2.7 Disconnected Property

Used to keep dataset opened after connection is closed.

Class

TCustomDADataSet

Syntax

```plaintext
property Disconnected: boolean;
```

Remarks

Set the Disconnected property to True to keep dataset opened after connection is closed.

© 1997-2021 Devart. All Rights Reserved.
5.11.1.5.2.8 FetchRows Property

Used to define the number of rows to be transferred across the network at the same time.

Class

TCustomDADataSet

Syntax

\textit{property} FetchRows: integer \textit{default} 25;

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.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.2.9 FilterSQL Property

Used to change the WHERE clause of SELECT statement and reopen a query.

Class

TCustomDADataSet

Syntax

\textit{property} FilterSQL: \textit{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.

\textbf{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

```properties
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

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 uses the metainformation sent by the server together with data.

**See Also**

- `SQLDelete`
- `SQLInsert`
- `SQLRefresh`
- `SQLUpdate`
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

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

```pascal
MyQuery.SQL.Text := 'SELECT * FROM Dept ORDER BY &Order';
MyQuery_MACROByName('Order').Value := 'DeptNo';
MyQuery.Open;
```

See Also
- TMacro
- MacroByName
- Params

5.11.1.5.2.15 MasterFields Property

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

```pascal
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.
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**

- **DetailFields**
- **MasterSource**
- **Master/Detail Relationships**
5.11.1.5.2.17 Options Property

Used to specify the behaviour of TCustomDADataSet object.

**Class**

**TCustomDADataSet**

**Syntax**

```plaintext
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><strong>AutoPrepare</strong></td>
<td>Used to execute automatic <a href="#">Prepare</a> on the query execution.</td>
</tr>
<tr>
<td><strong>CacheCalcFields</strong></td>
<td>Used to enable caching of the TField.Calculated and TField.Lookup fields.</td>
</tr>
<tr>
<td><strong>CompressBlobMode</strong></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>
</tbody>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th><strong>MasterFieldsNullable</strong></th>
<th>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).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NumberRange</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></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>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
Devar. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback

© 2021 Devar
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

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

© 1997-2021 Devart. All Rights Reserved.

Provide Feedback

5.11.1.5.2.19  ParamCount Property

Used to indicate how many parameters are there in the Params property.

Class

TCustomDADataSet

Syntax

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

```
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**

## 5.11.1.5.2.21  ReadOnly Property

Used to prevent users from updating, inserting, or deleting data in the dataset.

### Class
**TCustomDADataSet**

**Syntax**

```objectivec
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 RefreshOptions Property**

Used to indicate when the editing record is refreshed.

**Class**

TCustomDADataSet

**Syntax**

```objectivec
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

```delphi
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

```delphi
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.

5.11.5.2.25 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

```sql
DELETE FROM Orders
WHERE OrderID = :Old_OrderID
```

See Also
- SQL
- SQLInsert
- SQLUpdate
- SQLRefresh

© 1997-2021 Devart. All Rights Reserved.
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

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.

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

property SQLLock: TStrings;

Remarks

Use the SQLLock property to specify a SQL statement that will be used to perform a record lock.
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

| 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
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

```Delphi
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
```

See Also

- RefreshRecord
- SQL
- SQLInsert
- SQLUpdate
- SQLDelete
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

```
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.

To create a SQLUpdate statement at design-time, use the query statement editor.

Example

```
UPDATE Orders
  set  
    ShipName = :ShipName
WHERE
  OrderID = :Old_OrderID
```

See Also

- SQL
- SQLInsert
- SQLDelete
- SQLRefresh
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

- TMyQuery.FetchAll

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback

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>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 data to a BLOB field, specified by the Field parameter.</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>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</td>
<td>Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known. (inherited from TMemDataSet)</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 Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Locate</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>)</td>
</tr>
<tr>
<td><strong>LocateEx</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>)</td>
</tr>
<tr>
<td><strong>Lock</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MacroByName</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ParamByName</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Prepare</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RefreshRecord</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RestoreSQL</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RestoreUpdates</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>)</td>
</tr>
<tr>
<td><strong>RevertRecord</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>)</td>
</tr>
<tr>
<td><strong>SaveSQL</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SaveToXML</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>)</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SetRange</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>)</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.

### See Also
- `TCustomDADataSet Class`
- `TCustomDADataSet Class Members`

© 1997-2021

Devar. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.11.1.5.3.1  AddWhere Method

- Adds condition to the WHERE clause of SELECT statement in the SQL property.

**Class**

`TCustomDADataSet`
Syntax

```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`

5.11.1.5.3.2  BreakExec Method

Breaks execution of the SQL statement on the server.

Class

`TCustomDADataSet`

Syntax

```procedure BreakExec; virtual;```

Remarks

Call the `BreakExec` method to break execution of the SQL statement on the server. Execution is broken by the KILL operator execution on server. It makes sense to call `BreakExec` only from another thread.
### 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**

```pascal
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 the SQL property and assign BaseSQL.

See Also
- AddWhere
- BaseSQL

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(Iters: integer; Offset: integer)</td>
<td>Used to perform Batch operations.</td>
</tr>
</tbody>
</table>

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`

Used to perform Batch operations.

Class
`TCustomDADataSet`

Syntax

```plaintext
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
5.11.1.5.3.6 Executing Method

Indicates whether SQL statement is still being executed.

Class

TCustomDADataSet

Syntax

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.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.3.7 Fetched Method

Used to find out whether TCustomDADataSet has fetched all rows.

Class

TCustomDADataSet

Syntax

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
5.11.1.5.3.8 Fetching Method

Used to learn whether TCustomDADataSet is still fetching rows.

Class

TCustomDADataSet

Syntax

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

5.11.1.5.3.9 FetchingAll Method

Used to learn whether TCustomDADataSet is fetching all rows to the end.

Class

TCustomDADataSet

Syntax

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
5.11.5.3.10 FindKey Method

Searches for a record which contains specified field values.

**Class**

**TCustomDADataSet**

**Syntax**

```pascal
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.

5.11.5.3.11 FindMacro Method

Finds a macro with the specified name.

**Class**

**TCustomDADataSet**

**Syntax**

```pascal
function FindMacro(const Value: string): TMacro;
```

**Parameters**

*Value*

Holds the specified name.
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

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
TCustomDACDataSet

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.
5.11.1.5.3.13  FindParam Method

Determines if a parameter with the specified name exists in a dataset.

Class

TCustomDADataSet

Syntax

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
GetDataType Method

Returns internal field types defined in the MemData and accompanying modules.

Class

TCustomDADataSet

Syntax

function GetDataType(const FieldName: string): integer; virtual;

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.

© 1997-2021
Devart. All Rights Reserved.

GetFieldObject Method

Returns a multireference shared object from field.

Class

TCustomDADataSet

Syntax

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;

Parameters

(const FieldName: string): TSharedObject; overload;
**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**

```pascal
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

© 1997-2021
Devart. All Rights Reserved.
Request Support DAC Forum Provide Feedback

5.11.1.5.3.18 GetKeyFieldNames Method

Provides a list of available key field names.

Class

TCustomDADataSet

Syntax

procedure GetKeyFieldNames(List: TStrings);

Parameters

List

The list of available key field names

Return Value
GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

See Also
- TCustomDACConnection.GetTableNames
- TCustomDACConnection.GetStoredProcNames

Retrieves an ORDER BY clause from a SQL statement.

Class
TCustomDADataset

Syntax

```plaintext
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
5.11.1.5.3.20 GotoCurrent Method

Sets the current record in this dataset similar to the current record in another dataset.

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>. 
5.11.5.3.22 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.

Example

MyQuery.SQL := 'SELECT * FROM Scott.Dept ORDER BY &Order';
MyQuery.MacroByName('Order').Value := 'DeptNo';
MyQuery.Open;

See Also

• TMacro
5.11.1.5.3.23 ParamByName Method

Sets or uses parameter information for a specific parameter based on its name.

Class
TCustomDADataset

Syntax

\[
\text{function ParamByName(} \text{const Value: string): TDAParam;}
\]

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:

\[
\text{Edit1.Text := Query1.ParamsByName('Contact').AsString;}
\]

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 MySQL prepared protocol has certain server restrictions, and its work is not always stable. That is why it is advisable to perform test before using preparation in production versions of applications.

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

© 1997-2021 Devart. All Rights Reserved.

5.11.1.5.3.25 RefreshRecord Method

Actualizes field values for the current record.

Class

TCustomDADataSet

Syntax

```
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.

5.11.1.5.3.26 RestoreSQL Method

Restores the SQL property modified by AddWhere and SetOrderBy.

Class

TCustomDADataset

Syntax

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.
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

```plaintext
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

```plaintext
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
**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

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>(inherited from <strong>TMemDataSet</strong>) Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord</td>
<td>(inherited from <strong>TMemDataSet</strong>) Occurs when a single update component can not</td>
</tr>
</tbody>
</table>
handle the updates.

See Also
- TCustomDADataSet Class
- TCustomDADataSet Class Members

5.11.1.5.4.1 AfterExecute Event

Occurs after a component has executed a query to database.

Class
TCustomDADataSet

Syntax

```
property AfterExecute: TAfterExecuteEvent;
```

Remarks

Occurs after a component has executed a query to database.

See Also
- TCustomDADataSet.Execute

5.11.1.5.4.2 AfterFetch Event

Occurs after dataset finishes fetching data from server.

Class
TCustomDADataSet

Syntax

```
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

```pascal
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

```pascal
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.
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

```pascal
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>ParamCount</td>
<td>Indicates the number of parameters in the Params</td>
</tr>
<tr>
<td>Name</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.</td>
</tr>
</tbody>
</table>
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>

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 Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>Makes it possible to change</td>
</tr>
</tbody>
</table>
### ChangeCursor Property

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

```plaintext
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 MyDacVcl unit to the uses clause of any unit in your project to make the Debug property work.

**Note:** If TMySQLMonitor is used in the project and the TMySQLMonitor.Active property is set to False, the debug window is not displayed.

See Also

- TCustomDADataSet.Debug

© 1997-2021
Devart. All Rights Reserved.

5.11.1.6.2.4 FinalSQL Property

Used to return a SQL statement with expanded macros.

Class

TCustomDASQL

Syntax

```plaintext
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.

© 1997-2021
Devart. All Rights Reserved.
5.11.1.6.2.5  MacroCount Property

Used to get the number of macros associated with the Macros property.

Class

TCustomDASQL

Syntax

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

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
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

5.11.1.6.2.8 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.

### 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:

```delphi
procedure TForm1.Button1Click(Sender: TObject);
begin
  with MySQL 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;
```

© 1997-2021 Devart. All Rights Reserved.
5.11.1.6.2.10 ParamValues Property (Indexer)

Used to get or set the values of individual field parameters that are identified by name.

Class

TCustomDASQL

Syntax

\[
\text{property} \ \text{ParamValues}[\text{const ParamName: string}]: \text{Variant}; \text{default};
\]

Parameters

\text{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.
5.11.1.6.13 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

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>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</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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

5.11.1 BreakExec Method

Breaks execution of an SQL statement on the server.

Class
TCustomDASQL

Syntax

```plaintext
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.
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(Iters: integer; Offset: integer)</td>
<td>Used to perform <strong>Batch operations</strong>.</td>
</tr>
</tbody>
</table>

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 is ignored.
Used to perform **Batch operations**.

Class

**TCustomDASQL**

Syntax

```delphi
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**

© 1997-2021 Devart. All Rights Reserved.   Request Support   DAC Forum   Provide Feedback

5.11.1.6.3.3  Executing Method

Checks whether `TCustomDASQL` still executes a SQL statement.

Class

**TCustomDASQL**

Syntax

```delphi
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.

© 1997-2021 Devart. All Rights Reserved.

5.11.6.3.4 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.

5.11.6.3.5 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`

---

5.11.1.6.3.6 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.

5.11.1.3.7 `ParamByName` Method

Finds a parameter with the specified name.

**Class**

`TCustomDASQL`

**Syntax**

``` Pascal
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.

**Example**

``` Pascal
MyCommandSQL.Execute;
Edit1.Text := MyCommandSQL.ParamsByName('Contact').AsString;
```

**See Also**
5.11.6.8 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

TCustomDASQL

Syntax

```delphi
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 MySQL prepared protocol has certain server restrictions, and its work is not always stable. That is why it is advisable to perform test before using preparation in production versions of applications.

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

5.11.6.9 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.

See Also
- Executing

© 1997-2021 Devart. All Rights Reserved.

Request Support       DAC Forum       Provide Feedback
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

- TCustomDASQL Class
- TCustomDASQL Class Members

© 1997-2021 Devart. All Rights Reserved.

5.11.1.6.4.1 AfterExecute Event

Occurs after a SQL statement has been executed.

Class

TCustomDASQL

Syntax

```
property AfterExecute: TAfterExecuteEvent;
```

Remarks

Occurs after a SQL statement has been executed. This event may be used for descendant components which use multithreaded environment.

See Also

- TCustomDASQL.Execute

© 1997-2021 Devart. All Rights Reserved.
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

- TCustomMyDataSet.UpdateObject

© 1997-2021 Devart. All Rights Reserved.

5.11.1.7.1 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>
<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>
</tbody>
</table>
### InsertSQL
- **Used when inserting a record.**

### LockObject
- Provides ability to perform advanced adjustment of lock operations.

### LockSQL
- **Used to lock the current record.**

### ModifyObject
- Provides ability to perform advanced adjustment of modify operations.

### ModifySQL
- **Used when updating a record.**

### RefreshObject
- Provides ability to perform 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.

### SQL
- **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></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 TCustomDADataSet object that is being updated.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL 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 advanced adjustment of refresh operations.</td>
</tr>
<tr>
<td>RefreshSQL</td>
<td>Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataSet.RefreshRecord procedure.</td>
</tr>
</tbody>
</table>
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 TCustomMyDataSet 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
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.

5.11.1.7.2.4 InsertObject Property

Provides ability to perform advanced adjustment of insert operations.

Class
TCustomDAUpdateSQL

Syntax

```
property InsertObject: TComponent;
```

Remarks
Assign SQL component or TCustomMyDataSet 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`

Class
`TCustomDAUpdateSQL`

Syntax
```
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.

Class
`TCustomDAUpdateSQL`

Syntax
```
property LockObject: TComponent;
```

Remarks
Assign a SQL component or TCustomMyDataSet descendant to this property to perform advanced adjustment of lock operations.
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

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.

Class

TCustomDAUpdateSQL

Syntax

```property ModifyObject: TComponent;```

Remarks

Assign a SQL component or TCustomMyDataSet descendant to this property to perform advanced adjustment of modify operations.
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**

```property
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
property RefreshObject: TComponent;
```

**Remarks**

Assign a SQL component or TCustomMyDataSet 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

© 1997-2021
Devart. All Rights Reserved.

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

© 1997-2021
Devart. All Rights Reserved.

5.11.1.7.2.12 SQL Property(Indexer)

Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

© 2021 Devart
Class

**TCustomDAUpdateSQL**

Syntax

```pascal
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.

5.11.1.7.3 Methods

Methods 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><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>

See Also

- **TCustomDAUpdateSQL Class**
- **TCustomDAUpdateSQL Class Members**

© 1997-2021 Devart. All Rights Reserved.
5.11.1.7.3.1 Apply Method

Sets parameters for a SQL statement and executes it to update a record.

Class

TCustomDAUpdateSQL

Syntax

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

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 TDACondition Class**

Represents a condition from the **TDAConditions** list.

For a list of all members of this type, see **TDACondition** members.

**Unit**

**DBAccess**

**Syntax**

```
TDACondition = class(TCollectionItem);
```

**Remarks**

Manipulate conditions using **TDAConditions**.

**See Also**

- **TDAConditions**

© 1997-2021

Devart. All Rights Reserved.
5.11.1.8.1 Members

**TDACCondition** 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>

5.11.1.8.2 Properties

Properties of the **TDACCondition** class.

For a complete list of the **TDACCondition** class members, see the [TDACCondition 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>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

```property
Enabled: Boolean default True;
```

5.11.1.8.2.2  Name Property

The name of the condition

Class
TDACCondition

Syntax

```property
Name: string;
```

5.11.1.8.2.3  Value Property

The value of the condition

Class
TDACCondition

Syntax

```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.

5.11.1.8.3.1 Disable Method

Disables the condition

Class

**TDACCondition**

Syntax

```delphi
procedure Disable;
```

© 1997-2021 Devart. All Rights Reserved.

5.11.1.8.3.2 Enable Method

Enables the condition

Class

**TDACCondition**
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
(SAL<2500)
```

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><strong>Enabled</strong></td>
<td>Indicates whether the condition is enabled</td>
</tr>
<tr>
<td><strong>Items</strong></td>
<td>Used to iterate through all conditions.</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The property returns condition names and values as <code>CONDITION_NAME=CONDITION</code></td>
</tr>
<tr>
<td><strong>WhereSQL</strong></td>
<td>Returns the SQL WHERE condition added in the Conditions property.</td>
</tr>
</tbody>
</table>

**Methods**

| **Name** | **Description** |
| **Add** | Overloaded. Adds a condition to the WHERE clause of the query. |
| **Delete** | Deletes the condition |
| **Disable** | Disables the condition |
| **Enable** | Enables the condition |
| **Find** | Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil. |
| **Get** | Retrieving a TDACondition object by its name. If found, the TDACondition object is returned, otherwise - an exception is raised. |
| **IndexOf** | Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1. |
| **Remove** | Removes the condition |
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 <strong>CONDITION_NAME=COND</strong>ITION</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](#)

© 1997-2021 Devart. All Rights Reserved.

Used to iterate through all the conditions.

**Class**

**TDAConditions**

**Syntax**

```pascal
property Condition[Index: Integer]: TDACondition;
```
5.11.1.9.2.2  Enabled Property

Indicates whether the condition is enabled

Class

TDAConditions

Syntax

```plaintext
property Enabled: Boolean;
```

5.11.1.9.2.3  Items Property(Indexer)

Used to iterate through all conditions.

Class

TDAConditions

Syntax

```plaintext
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; |

5.11.1.9.3 Methods

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>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>
</tbody>
</table>
### Enable
Enables the condition

### Find
Search for TDACCondition (the condition) by its name. If found, the TDACCondition object is returned, otherwise - nil.

### Get
Retrieving a TDACCondition object by its name. If found, the TDACCondition object is returned, otherwise - an exception is raised.

### IndexOf
Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.

### Remove
Removes the condition

### See Also
- TDACConditions Class
- TDACConditions Class Members

© 1997-2021
Devart. All Rights Reserved.

See Also
- TDACConditions Class
- TDACConditions Class Members

© 1997-2021
Devart. All Rights Reserved.

### Add Method

 Adds a condition to the WHERE clause of the query.

### Class
**TDACConditions**

### 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>

5.11.1.9.3.1 Add Method

© 2021 Devart
Reserved.

Adds a condition to the WHERE clause of the query.

Class

**TDAConditions**

**Syntax**

```
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:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
    and
    (SAL<2500)
```

Adds a condition to the WHERE clause of the query.

Class

**TDAConditions**

**Syntax**

```
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**

```
procedure Delete(Index: integer);
```

**Parameters**

**Index**
Index of the condition

### 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

**TDACConditions**

Syntax

```pascal
procedure Enable;
```

5.11.1.9.3.5 Find Method

Search for TDACCondition (the condition) by its name. If found, the TDACCondition object is returned, otherwise - nil.

Class

**TDACConditions**

Syntax

```pascal
function Find(const Name: string): TDACCondition;
```

Parameters

- **Name**

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**
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
TDAConditions

Syntax

```python
function IndexOf(const Name: string): Integer;
```

Parameters

Name

5.11.1.9.3.8 Remove Method

Removes the condition

Class
TDAConditions

Syntax

```python
procedure Remove(const Name: string);
```

Parameters

Name

Specifies the name of the removed condition
5.11.1.10 TDACConnectionOptions Class

This class allows setting up the behaviour of the TDACConnection class.

For a list of all members of this type, see TDACConnectionOptions members.

Unit

DBAccess

Syntax

TDACConnectionOptions = class(TPersistent);

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>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</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------</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 TCustomDACConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
</tbody>
</table>
### Published

<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>
</tbody>
</table>

### See Also
- [TDAConnectionOptions Class](#)
- [TDAConnectionOptions Class Members](#)

5.11.1.10.2.1 Allow ImplicitConnect Property

Specifies whether to allow or not implicit connection opening.

### Class

**TDAConnectionOptions**

### Syntax

```plaintext
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.

5.11.1.10.2.3 DisconnectedMode Property

Used to open a connection only when needed for performing a server call and closes after performing the operation.

Class

`TDACConnectionOptions`

Syntax

```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.
5.11.1.10.2.4 KeepDesignConnected Property

Used to prevent an application from establishing a connection at the time of startup.

Class
TDACollectionOptions

Syntax

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.

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
TDACollectionOptions

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.
5.11.1.11 TDACConnectionSSLOptions Class

This class is used to set up the SSL options.

For a list of all members of this type, see TDACConnectionSSLOptions members.

Unit

DBAccess

Syntax

TDACConnectionSSLOptions = class(TPersistent);

Properties

<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>

© 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
- TDAConnectionSSLOptions Class
- TDAConnectionSSLOptions Class Members

5.11.1.11.2.1  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.

5.11.1.11.2.2  Cert Property

Holds the path to the client certificate.

Class
**TDACConnectionSSLOptions**

**Syntax**

```plaintext
property Cert: string;
```

**Remarks**

Use the Cert property to specify the path to the client certificate.

---

5.11.1.11.2.3 CipherList Property

**Remarks**

Holds the list of allowed SSL ciphers.

**Class**

TDACConnectionSSLOptions

**Syntax**

```plaintext
property CipherList: string;
```

**Remarks**

Use the CipherList property to specify the list of allowed SSL ciphers.

---

5.11.1.11.2.4 Key Property

**Remarks**

Holds the path to the private client key.

**Class**

TDACConnectionSSLOptions

**Syntax**

```plaintext
property Key: string;
```

**Remarks**
Use the Key property to specify the path to the private client key.

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

TDADataSetOptions = class(TPersistent);  

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPrepare</td>
<td>Used to execute automatic TCustomDADDataSet.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</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FieldsOrigin</td>
<td>Refreshing detail dataset while navigating master dataset.</td>
</tr>
<tr>
<td>FlatBuffers</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>InsertAllSetFields</td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td>LocalMasterDetail</td>
<td>Used to include all set dataset fields in the generated INSERT statement.</td>
</tr>
<tr>
<td>MasterFieldsNullable</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>NumberRange</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>QueryRecCount</td>
<td>Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.</td>
</tr>
<tr>
<td></td>
<td>Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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>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>
Properties of the `TDADatasetOptions` class.

For a complete list of the `TDADatasetOptions` class members, see the [TDADatasetOptions Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoPrepare</strong></td>
<td>Used to execute automatic <code>TCustomDADataSet.Prepare</code> on the query execution.</td>
</tr>
<tr>
<td><strong>CacheCalcFields</strong></td>
<td>Used to enable caching of the <code>TField.Calculated</code> and <code>TField.Lookup</code> fields.</td>
</tr>
<tr>
<td><strong>CompressBlobMode</strong></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 <code>DefaultExpression</code> 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 <code>TCustomDADataSet</code> to fill the <code>Origin</code> property of the <code>TField</code> 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 <code>ftString</code> and <code>ftVarBytes</code> 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 <code>TCustomDADataSet</code> to use</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Local Filtering</td>
<td>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>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>
</tbody>
</table>
**SetFieldsReadOnly**

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**

Used for `TCustomDADataSet` to raise an exception when the number of updated or deleted records is not equal 1.

**UpdateAllFields**

Used to include all dataset fields in the generated `UPDATE` and `INSERT` statements.

**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](#)

© 1997-2021 Devart. All Rights Reserved.
query execution. Makes sense for cases when a query will be executed several times, for example, in Master/Detail relationships.

5.11.1.12.2.2 CacheCalcFields Property

Used to enable caching of the TField.Calculated and TField.Lookup fields.

Class

TDADatasetOptions

Syntax

```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.

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.
5.11.12.2.4 DefaultValues Property

Used to request default values/expressions from the server and assign them to the DefaultExpression property.

Class

TDADatasetOptions

Syntax

```plaintext
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.

5.11.12.2.5 DetailDelay Property

Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.

Class

TDADatasetOptions

Syntax

```plaintext
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

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

`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.

© 1997-2021 Devart. All Rights Reserved.

---

5.11.1.12.2.13  QueryRecCount Property

Used for `TCustomDAODataSet` 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, `TCustomDAODataSet` 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 `TCustomDAODataSet` to quote all database object names in autogenerated SQL statements such as update SQL.

### Class

**TDADatasetOptions**

© 2021 Devart
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 TCustomDADDataSet to set the Required property of the TField objects for the NOT NULL fields.

Class

TDADataSetOptions
### RequiredFields Property

**Syntax**

```delphi
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.

### ReturnParams Property

**Syntax**

```delphi
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.

### SetFieldsReadOnly Property

**Syntax**

```delphi
property SetFieldsReadOnly: boolean default False;
```

**Remarks**

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.
**SetFieldsReadOnly**

```
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 cannot be updated. Set this option for datasets that use automatic generation of the update SQL statements only.

---

**StrictUpdate Property**

Used for TCustomDADataset to raise an exception when the number of updated or deleted records is not equal 1.

**Class**

`TDADatasetOptions`

**Syntax**

```
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.

---

**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.

**5.11.1.12.2.21 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.
Syntax

```plaintext
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>

### 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>
5.11.13.1.2.1 Encryptor Property

Used to specify the encryptor class that will perform the data encryption.

Class

TDAEncryption

Syntax

```delphi
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.

5.11.13.2.2 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>
</table>

## Properties

Properties of the `TDAMapRule` class.

For a complete list of the `TDAMapRule` class members, see the [TDAMapRule Members](#) topic.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DBLengthMax</code></td>
<td>Maximum DB field length, until which the rule is applied.</td>
</tr>
<tr>
<td><code>DBLengthMin</code></td>
<td>Minimum DB field length, starting from which the rule is applied.</td>
</tr>
<tr>
<td><code>DBScaleMax</code></td>
<td>Maximum DB field scale, until which the rule is applied to the specified DB field.</td>
</tr>
<tr>
<td><code>DBScaleMin</code></td>
<td>Minimum DB field Scale, starting from which the rule is applied to the specified DB field.</td>
</tr>
<tr>
<td><code>DBType</code></td>
<td>DB field type, that the rule is applied to.</td>
</tr>
<tr>
<td><code>FieldName</code></td>
<td>DataSet field name, for which the rule is applied.</td>
</tr>
<tr>
<td><code>FieldScale</code></td>
<td>The resultant field Scale in Delphi.</td>
</tr>
<tr>
<td><code>FieldType</code></td>
<td>Delphi field type, that the specified DB type or DataSet field will be mapped to.</td>
</tr>
<tr>
<td><code>IgnoreErrors</code></td>
<td>Ignoring errors when converting data from DB to Delphi type.</td>
</tr>
</tbody>
</table>
### DBLengthMax Property

Maximum DB field length, until which the rule is applied.

#### Class

**TDAMapRule**
**property** DBLengthMax: Integer **default** rlAny;

Remarks

Setting maximum DB field length, until which the rule is applied to the specified DB field.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.14.2.2 DBLengthMin Property

Minimum DB field length, starting from which the rule is applied.

Class

TDAMapRule

Syntax

**property** DBLengthMin: Integer **default** rlAny;

Remarks

Setting minimum DB field length, starting from which the rule is applied to the specified DB field.

© 1997-2021
Devart. All Rights Reserved.

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

Syntax

**property** DBScaleMax: Integer **default** rlAny;

Remarks

Setting maximum DB field scale, until which the rule is applied to the specified DB field.

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

5.11.1.14.2.4 DBScaleMin Property

Minimum DB field Scale, starting from which the rule is applied to the specified DB field.

Class
TDAMapRule

Syntax

```pascal
property DBScaleMin: Integer default rlAny;
```

Remarks
Setting minimum DB field Scale, starting from which the rule is applied to the specified DB field.

5.11.1.14.2.5 DBType Property

DB field type, that the rule is applied to.

Class
TDAMapRule

Syntax

```pascal
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 rule will be applied to all fields of the specified type in all DataSets related to this Connection.
5.11.1.14.2.6 FieldLength Property

The resultant field length in Delphi.

Class
TDAMapRule

Syntax

```
property FieldLength: Integer default rlAny;
```

Remarks

Setting the Delphi field length after conversion.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.14.2.7 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.

© 1997-2021
Devart. All Rights Reserved.

5.11.1.14.2.8 FieldScale Property

The resultant field Scale in Delphi.

Class

© 2021 Devart
**TDAMapRule**

### Syntax

```property```
property FieldScale: Integer `default` rlAny;
```property```

### Remarks

Setting the Delphi field Scale after conversion.

---

**5.11.14.2.9  FieldType Property**

Delphi field type, that the specified DB type or DataSet field will be mapped to.

**Class**

**TDAMapRule**

### Syntax

```property```
property FieldType: TFieldType `stored` IsFieldType Stored `default` ftUnknown;
```property```

### Remarks

Setting Delphi field type, that the specified DB type or DataSet field will be mapped to.

---

**5.11.14.2.10  IgnoreErrors Property**

Ignoring errors when converting data from DB to Delphi type.

**Class**

**TDAMapRule**

### Syntax

```property```
property IgnoreErrors: Boolean `default` False;
```property```

### 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.

5.11.1.15 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

5.11.1.15.1  Members

TDAMapRules class overview.

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>

© 1997-2021 Devart. All Rights Reserved.
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**](#)

© 1997-2021 Devart. All Rights Reserved.

5.11.1.15.2.1  IgnoreInvalidRules Property

Used to avoid raising exception on mapping rules that can't be applied.

**Class**

**TDAMapRules**

**Syntax**

```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**](#)
5.11.16 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:

```pascal
MetaData.Connection := Connection;
MetaData.MetaDataKind := 'Columns';
MetaData.Restrictions.Values['TABLE_NAME'] := 'Emp';
MetaData.Open;
```

Inheritance Hierarchy

```
TMemDataSet
  TDAMetaData
```

See Also
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>Connection</td>
<td>Used to specify a connection object to use to connect to a data store.</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>MetaDataKind</td>
<td>Used to specify which kind of metainformation to show.</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>Restrictions</td>
<td>Used to provide one or more conditions restricting the list.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</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>

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></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>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>Prepare</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Allocates resources and creates field components for a dataset.</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>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>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...</td>
</tr>
</tbody>
</table>
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>

5.11.1.16.2 Properties

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><strong>CachedUpdates</strong></td>
<td>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>Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong></td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>Used to avoid setting the</td>
</tr>
</tbody>
</table>
Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

**LocalUpdate** (inherited from **TMemDataSet**)  
Used to prevent implicit update of rows on database server.

**MetaDataKind**  
Used to specify which kind of metainformation to show.

**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.

**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.

**See Also**
- **TDAMetaData Class**
- **TDAMetaData Class Members**

© 1997-2021 Devart. All Rights Reserved.

Request Support  
DAC Forum  
Provide Feedback

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**;

```property```

**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</td>
</tr>
<tr>
<td>ProcedureParameters</td>
<td>show metainformation about parameters of existing procedures</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</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

Class
TDArea

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

Methods

Methods of the TDArea class.

For a complete list of the TDArea class members, see the TDArea Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

5.11.1.16.3 Methods

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

5.11.1.16.2.3 Restrictions Property

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Inherited From</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyRange</td>
<td>TMemDataSet</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td>ApplyUpdates</td>
<td>TMemDataSet</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td>CancelRange</td>
<td>TMemDataSet</td>
<td>Removes any ranges currently in effect for a dataset.</td>
</tr>
<tr>
<td>CancelUpdates</td>
<td>TMemDataSet</td>
<td>Clears all pending cached updates from cache and restores dataset in its prior state.</td>
</tr>
<tr>
<td>CommitUpdates</td>
<td>TMemDataSet</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>DeferredPost</td>
<td>TMemDataSet</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>EditRangeEnd</td>
<td>TMemDataSet</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart</td>
<td>TMemDataSet</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>GetBlob</td>
<td>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>GetMetaDataKinds</td>
<td></td>
<td>Used to get values acceptable in the MetaDataKind property.</td>
</tr>
<tr>
<td>GetRestrictions</td>
<td></td>
<td>Used to find out which restrictions are applicable to a certain MetaDataKind.</td>
</tr>
<tr>
<td>Locate</td>
<td>TMemDataSet</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>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</td>
<td>TMemDataSet</td>
<td>Allocates resources and creates field components for</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td></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>
<td></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>
<td></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>
<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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td><strong>UpdateResult</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Reads the status of the latest call to the <code>ApplyUpdates</code> method while cached updates are enabled.</td>
<td></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>
<td></td>
</tr>
</tbody>
</table>

**See Also**
- **TDAMetaData Class**
- **TDAMetaData Class Members**

© 2021 Devart
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`
- `List`
Holds the object that will be filled with metadata kinds (restrictions).

MetaDataSetKind
Holds the metadata kind for which restrictions are returned.

Remarks
Call the GetRestrictions method to find out which restrictions are applicable to a certain MetaDataSetKind. The List parameter will be cleared and then filled with values.

See Also
• Restrictions
• GetMetaDataSetKinds

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

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
### TDAParam 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 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</td>
</tr>
<tr>
<td></td>
<td>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>
</tbody>
</table>
### Properties

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><code>AsBlob</code></td>
<td>Used to set and read the</td>
</tr>
</tbody>
</table>
### 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>
</tbody>
</table>
### Value

| Value         | Used to represent the value of the parameter as Variant. |

#### 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

```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.

5.11.1.17.2.5 AsLargeInt Property

Used to assign the value for a LargeInteger field to the parameter.

Class
TDAParam

Syntax

```
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.

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.

Class

TDAParam

Syntax

```pascal
property AsMemoRef: TBlob;
```

Remarks

Use the AsMemoRef property to set and read the value of the memo parameter as a TBlob object.

5.11.17.2.7 AsMemoRef Property

Used to set and read the value of the memo parameter as a TBlob object.

Class

TDAParam

Syntax

```pascal
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.

5.11.17.2.8 AsSQLTimeStamp Property

Used to specify the value of the parameter when it represents a SQL timestamp field.
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

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

```objectivec
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

```objectivec
property 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

```
property 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.

5.11.1.17.2.14  Size Property

Specifies the size of a string type parameter.

Class

TDAParam

Syntax

```
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.
5.11.1.17.2.15 Value Property

Used to represent the value of the parameter as Variant.

Class

**TDAParam**

Syntax

```pascal
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.

Request Support DAC Forum Provide Feedback

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.

Public

<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>

© 2021 Devart
5.11.1.17.3.1 AssignField Method

Assigns field name and field value to a param.

Class

TDAParam

Syntax

```plaintext
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

```plaintext
procedure AssignFieldValue(Field: TField; const Value: Variant);
virtual;
```

Parameters

Field

Holds the field which name and value should be assigned to the param.
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.

5.11.1.17.3.3 LoadFromFile Method

Places the content of a specified file into a TDAParam object.

**Class**
TDAParam

**Syntax**

```pascal
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
5.11.1.17.3.4 LoadFromStream Method

Places the content from a stream into a TDAParam object.

Class

TDAParam

Syntax

procedure LoadFromStream(Stream: TStream; BlobType: TBlobType);
virtual;

Parameters

Stream
Holds the stream to copy content from.

BlobType
Holds a value that modifies the DataType property so that this TDAParam object now holds the BLOB value.

Remarks

Call the LoadFromStream method to place the content from a stream into a TDAParam object. The BlobType value modifies the DataType property so that this TDAParam object now holds the BLOB value.

See Also

- LoadFromFile

© 1997-2021 Devart. All Rights Reserved.

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>
</tbody>
</table>
SetBlobData(Buffer: IntPtr; Size: Integer)

Writes the data from a specified buffer to BLOB.

© 1997-2021 Devart. All Rights Reserved.

Parameters

Buffer
- Holds the pointer to the 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

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

© 1997-2021 Devart. All Rights Reserved.

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>
</table>
5.11.18.2  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

5.11.18.2.1  Items Property(Indexer)

Used to iterate through all parameters.

Class
TDAParams

Syntax

```pascal
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.

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.
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.

© 1997-2021 Devart. All Rights Reserved.

---

5.11.1.18.3.2  **ParamByName Method**

Searches for a parameter with the specified name.

**Class**

**TDAParams**

**Syntax**

```delphi
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 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
### 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</td>
</tr>
</tbody>
</table>
### Active Property

Used to determine if the transaction is active.

**Class**

**TDATransaction**

**Syntax**

```plaintext
property Active: boolean;
```

**Remarks**

Indicates whether the transaction is active. This property is read-only.

### 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**

```plaintext

```
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.

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

Commit Method

Commits the current transaction.

Class

TDATransaction
Syntax

```plaintext
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

5.11.19.3.2 Rollback Method

Discards all modifications of data associated with the current transaction and ends the transaction.

Class

`TDATransaction`

Syntax

```plaintext
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. Request Support DAC Forum Provide Feedback

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>
</tbody>
</table>
### OnCommit Retaining

Occurs after CommitRetaining has been executed.

### OnError

Used to process errors that occur during executing a transaction.

### OnRollback

Occurs after the transaction has been successfully rolled back.

### OnRollback Retaining

Occurs after RollbackRetaining has been executed.

#### See Also
- [TDATransaction Class](#)
- [TDATransaction Class Members](#)

---

#### 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.MyDac.TMyTransaction.CommitRetaining() method execution, the OnCommitRetaining event is used. When an error occurs during commit, the OnError event fires.

#### See Also
- [Commit](#)
- [OnError](#)
5.11.1.19.4.2 OnCommitRetaining Event

Occurs after CommitRetaining has been executed.

Class

TDATransaction

Syntax

```plaintext
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

```plaintext
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

```delphi
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.MyDac.TMyTransaction.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

```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**

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

```pascal
MyQuery.SQL.Text := 'SELECT * FROM Dept WHERE DeptNo > 20 &Cond1';
MyQuery.Macros[0].Value := 'and DName is NULL';
MyQuery.Macros[0].Active := False;
```

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.

AsFloat Property

Used to set the float value to a macro.

Class

TMacro

Syntax

```
property AsFloat: double;
```

Remarks
Use the AsFloat property to set the float value to a macro.

**Class**

**TMacro**

**Syntax**

```property```
```
property AsFloat: float;
```

**Remarks**

Use the AsFloat property to set the float value to a macro.

5.11.1.20.2.4 AsInteger Property

Used to set the integer value to a macro.

**Class**

**TMacro**

**Syntax**

```property```
```
property AsInteger: integer;
```

**Remarks**

Use the AsInteger property to set the integer value to a macro.

5.11.1.20.2.5 AsString Property

Used to assign the string value to a macro.

**Class**

**TMacro**

**Syntax**

```property```
```
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.
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.

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.
**DBAccess**

**Syntax**

```pascal
TMacros = class(TCollection);
```

**Remarks**

Use TMacros to manage a list of TMacro objects for the TCustomDASQL or TCustomDADataSet components.

**See Also**

- TMacro

© 1997-2021 Devart. All Rights Reserved.

© 2021 Devart

### 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 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</td>
</tr>
</tbody>
</table>
5.11.1.21.2 Properties

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>Items</td>
<td>Used to iterate through all the macros parameters.</td>
</tr>
</tbody>
</table>

See Also
- TMacros Class
- TMacros Class Members

5.11.1.21.2.1 Items Property(Indexer)

Used to iterate through all the macros parameters.

Class

TMacros

Syntax

```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.
5.11.1.21.3  Methods

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](#)

Copies the macros values and properties from the specified source.

**Class**

**TMacros**

**Syntax**
**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.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.21.3.2 Expand Method

Changes the macros in the passed SQL statement to their values.

**Class**

TMacros

**Syntax**

**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.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.21.3.3 FindMacro Method

Finds a macro with the specified name.

**Class**

TMacros

**Syntax**
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.

5.11.1.21.3.4 IsEqual Method

Compares itself with another TMacro object.

Class
TMacros

Syntax

function IsEqual(Value: TMacros): boolean;

Parameters

Value
Holds the values of TMacro objects.

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.

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
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

procedure Scan(const SQL: string);
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

```
TPoolingOptions = class(TPersistent);
```

5.11.1.22.1 Members

TPoolingOptions class overview.

Properties

<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</td>
</tr>
</tbody>
</table>
### 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><code>ConnectionLifetime</code></td>
<td>Used to specify the maximum time during which an opened connection can be used by connection pool.</td>
</tr>
<tr>
<td><code>MaxPoolSize</code></td>
<td>Used to specify the maximum number of connections that can be opened in connection pool.</td>
</tr>
<tr>
<td><code>MinPoolSize</code></td>
<td>Used to specify the minimum number of connections that can be opened in the connection pool.</td>
</tr>
<tr>
<td><code>Validate</code></td>
<td>Used for a connection to be validated when it is returned from the pool.</td>
</tr>
</tbody>
</table>

**See Also**

- [TPoolingOptions Class](#)
- [TPoolingOptions Class Members](#)
5.11.1.22.2.1 ConnectionLifetime Property

Used to specify the maximum time during which an opened connection can be used by connection pool.

Class

TPoolingOptions

Syntax

```pascal
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.

© 1997-2021 Devart. All Rights Reserved.

5.11.1.22.2.2 MaxPoolSize Property

Used to specify the maximum number of connections that can be opened in connection pool.

Class

TPoolingOptions

Syntax

```pascal
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.

© 1997-2021 Devart. All Rights Reserved.

© 2021 Devart
5.11.1.22.2.3 MinPoolSize Property

Used to specify the minimum number of connections that can be opened in the connection pool.

Class

TPoolingOptions

Syntax

```
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

```
property Validate: boolean default DefValValidate;
```

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);
```

5.11.1.23.1 Members

**TSmartFetchOptions** class overview.

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](#)

© 1997-2021 Devart. All Rights Reserved.

### 5.11.1.23.2.1 Enabled Property

Sets SmartFetch mode enabled or not.

**Class**

[TSmartFetchOptions](#)

**Syntax**

```property
property Enabled: Boolean default False;
```

© 1997-2021 Devart. All Rights Reserved.

### 5.11.1.23.2.2 LiveBlock Property

Used to minimize memory consumption.

**Class**

[TSmartFetchOptions](#)
### LiveBlock Property

**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 `FetchRows` property, 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.

---

### 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.

---

### SQLGetKeyValues Property

SQL query for the read key and prefetched fields from the database.

**Class**

`TSmartFetchOptions`
Syntax

```
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.

<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>TCustomDACConnection.OnConnectionLost</code> event.</td>
</tr>
<tr>
<td>TDAConectionErrorEvent</td>
<td>This type is used for the <code>TCustomDACConnectionOnError</code> event.</td>
</tr>
<tr>
<td>TDATransactionErrorEvent</td>
<td>This type is used for the <code>TDATransactionOnError</code> event.</td>
</tr>
<tr>
<td>TRefreshOptions</td>
<td>Represents the set of <code>TRefreshOption</code>.</td>
</tr>
<tr>
<td>TUpdateExecuteEvent</td>
<td>This type is used for the <code>TCustomDADataset.AfterUpdateExecute</code> and</td>
</tr>
</tbody>
</table>
5.11.2.1 TAfterExecuteEvent Procedure Reference

This type is used for the TCustomDADataSet.AfterExecute and TCustomDASQL.AfterExecute events.

Unit
DBAccess

Syntax

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 TCustomDACConnection.OnConnectionLost event.

Unit

DBAccess

Syntax

TConnectionLostEvent = procedure (Sender: TObject; Component: TComponent; ConnLostCause: TConnLostCause; var RetryMode: TRetryMode) of object;

Parameters

Sender
An object that raised the event.

Component

ConnLostCause
The reason of the connection loss.
5.11.2.5 TDAConnectionErrorEvent Procedure Reference

This type is used for the TCustomDACConnection.OnError event.

Unit

DBAccess

Syntax

TDAConnectionErrorEvent = procedure (Sender: TObject; E: EDAError; var Fail: boolean) of object;

Parameters

Sender
An object that raised the event.

E
The error information.

Fail
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;
An object that raised the event.

\( E \)
The error code.

\textit{Fail}
False, if an error dialog should be prevented from being displayed and \texttt{EAbort} exception to cancel the current operation should be raised.

5.11.2.7 \textbf{TRefreshOptions Set}

Represents the set of \texttt{TRefreshOption}.

\textbf{Unit}
\texttt{DBAccess}

\textbf{Syntax}
\begin{verbatim}
TRefreshOptions = set of TRefreshOption;
\end{verbatim}

5.11.2.8 \textbf{TUpdateExecuteEvent Procedure Reference}

This type is used for the \texttt{TCustomDADataSet.AfterUpdateExecute} and \texttt{TCustomDADataSet.BeforeUpdateExecute} events.

\textbf{Unit}
\texttt{DBAccess}

\textbf{Syntax}
\begin{verbatim}
TUpdateExecuteEvent = procedure (Sender: TDataSet; StatementTypes: TStatementTypes; Params: TDAParams) of object;
\end{verbatim}

\textbf{Parameters}
\texttt{Sender}
An object that raised the event.

\texttt{StatementTypes}
Holds the type of the SQL statement being executed.

\texttt{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>

5.11.3.1 TLabelSet Enumeration

Sets the language of labels in the connect dialog.

Unit

DBAccess

Syntax

```pascal
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>
</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>No locking occurs. This mode should only be used in single user applications. The default value.</td>
</tr>
<tr>
<td>lmOptimistic</td>
<td>Locking occurs when the user posts an edited record, then the lock is released. Locking is done by the RefreshRecord method.</td>
</tr>
<tr>
<td>lmPessimistic</td>
<td>Locking occurs when the user starts editing a record. The lock is released after the user has posted or canceled the changes.</td>
</tr>
</tbody>
</table>

5.11.3.3 TRefreshOption Enumeration

Indicates when the editing record will be refreshed.

Unit

DBAccess
Syntax

\[ \text{TRefreshOption} = (\text{roAfterInsert}, \text{roAfterUpdate}, \text{roBeforeEdit}); \]

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>

5.11.3.4 TRetryMode Enumeration

Specifies the application behavior when connection is lost.

Unit

DBAccess

Syntax

\[ \text{TRetryMode} = (\text{rmRaise}, \text{rmReconnect}, \text{rmReconnectExecute}); \]

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>rmReconnectExecute</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.

Variables
<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 BaseSQL 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**

```plaintext
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 MyDAC 4.00.2.8. 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, MyDAC 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 MyDAC 5.00.0.4. To restore the old behavior, set the BaseSQLOldBehavior variable to True.
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 MyDAC.</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>

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
### 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 MyDAC.</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>
</tbody>
</table>
## TObjectType

This class is not used.

## TSharedObject

A base class that allows to simplify memory management for object referenced by several other objects.

### 5.12.1.1 TAttribute Class

TAttribute is not used in MyDAC.

For a list of all members of this type, see `TAttribute` members.

**Unit**

`MemData`

**Syntax**

```delphi
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</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<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>
</tbody>
</table>

5.12.1.1.2 Properties

Properties of the TAttribute class.

For a complete list of the TAttribute class members, see the TAttribute Members topic.
### AttributeNo Property

Returns an attribute's ordinal position in object.

#### Class

**TAttribute**

#### Syntax

```pascal
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

```pascal
property DataSize: Integer;
```

Remarks
Use the DataSize property to learn the size of an attribute value in internal representation.

5.12.1.2.3 DataType Property

Returns the type of data that was assigned to the Attribute.

Class
TAttribute

Syntax

```pascal
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.

5.12.1.2.4 Length Property

Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.

Class
TAttribute

Syntax

```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

```property`` ObjectType: TObjectType;
```

Remarks

Use the ObjectType property to return a TObjectType object for an object attribute.
### Owner Property

Indicates TObjectType that uses the attribute to represent one of its attributes.

**Class**

`TAttribute`

**Syntax**

```pascal
property Owner: TObjectType;
```

**Remarks**

Check the value of the Owner property to determine TObjectType that uses the attribute to represent one of its attributes. Applications should not assign the Owner property directly.

### 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**
### 5.12.1.2.9 Size Property

Returns the size of an attribute value in external representation.

**Class**

*TAttribute*

**Syntax**

```delphi
property Size: Integer;
```

**Remarks**

Read Size to learn the size of an attribute value in external representation.

For example:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtDate</td>
<td>8</td>
</tr>
<tr>
<td>dtFloat</td>
<td>8</td>
</tr>
<tr>
<td>dtInteger</td>
<td>4</td>
</tr>
</tbody>
</table>

(sizeof(TDateTime)

See Also

- [DataSize](#)

### 5.12.1.2 TBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types.

For a list of all members of this type, see TBlob members.

**Unit**

*MemData*
Syntax

```delphi
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
- `TMemDataSet.GetBlob`

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AsString</code></td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td><code>AsWideString</code></td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td><code>IsUnicode</code></td>
<td>Gives choice of making TBlob store and process data in Unicode format or not.</td>
</tr>
<tr>
<td><code>RefCount</code></td>
<td>Used to return the count of reference to a TSharedObject object.</td>
</tr>
<tr>
<td><code>Size</code></td>
<td>Used to learn the size of the TBlob value in bytes.</td>
</tr>
</tbody>
</table>

Methods
## Properties

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><code>AsString</code></td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td><code>AsWideString</code></td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
</tbody>
</table>
### AsString Property

Used to manipulate BLOB value as string.

**Class**

TBlob

**Syntax**

```pascal
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

 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.
5.12.1.2.4 Size Property

Used to learn the size of the TBlob value in bytes.

Class

TBlob

Syntax

```pascal
property size: Cardinal;
```

Remarks

Use the Size property to find out the size of the TBlob value in bytes.

© 1997-2021 Devart. All Rights Reserved.

5.12.1.2.3 Methods

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</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>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</td>
<td>(inherited from TSharedObject) Decrements the reference count.</td>
</tr>
</tbody>
</table>
### SaveToFile
Saves the contents of the `TBlob` object to a file.

### SaveToStream
Copies the contents of a `TBlob` object to a stream.

### Truncate
Sets new `TBlob` size and discards all data over it.

### Write
Stores a raw sequence of bytes into a `TBlob` object.

### See Also
- `TBlob Class`
- `TBlob Class Members`

```sql
5.12.1.2.3.1  Assign Method
```

Sets BLOB value from another `TBlob` object.

### Class

`TBlob`

### Syntax

```sql
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

```plaintext
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

5.12.1.2.3.5 Read Method

Acquires a raw sequence of bytes from the data stored in TBlob.

Class

TBlob

Syntax

```plaintext
function Read(Position: Cardinal; Count: Cardinal; Dest: IntPtr): Cardinal;
virtual;
```

Parameters

Position

Holds the starting point of the byte sequence.

Count

© 1997-2021 Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback
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**

```plaintext
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.

Stores a raw sequence of bytes into a TBlob object.

Class
TBlob

Syntax

\[
\text{procedure write(Position: Cardinal; Count: Cardinal; Source: IntPtr); virtual;}
\]

Parameters

- \text{Position}
  - Holds the starting point of the byte sequence.

- \text{Count}
  - Holds the size of the sequence in bytes.

- \text{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
- \text{Read}
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

MemoData

Syntax

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
var
    CompressProc: TCompressProc;
    UncompressProc: TUncompressProc;
```

Inheritance Hierarchy

```
TSharedObject
  TBlob
    TCompressedBlob
```

See Also

- TBlob
- TMemDataSet.GetBlob
### TCompressedBlob class overview.

#### 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>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>(inherited from TBlob) Gives choice of making TBlob store and process data in Unicode format or not.</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>(inherited from TBlob) 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</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 TBlob) Sets BLOB value from another TBlob object.</td>
</tr>
<tr>
<td>Clear</td>
<td>(inherited from TBlob) Deletes the current value in TBlob object.</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>(inherited from TBlob) Loads the contents of a file into a TBlob object.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------</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>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>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>Write</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Stores a raw sequence of bytes into a TBlob object.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

### 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><strong>AsString</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Used to manipulate BLOB value as string.</td>
</tr>
<tr>
<td><strong>AsWideString</strong> (inherited from <strong>TBlob</strong>)</td>
<td>Used to manipulate BLOB value as Unicode string.</td>
</tr>
<tr>
<td><strong>Compressed</strong></td>
<td>Used to indicate if the Blob is compressed.</td>
</tr>
<tr>
<td><strong>CompressedSize</strong></td>
<td>Used to indicate compressed size of the Blob data.</td>
</tr>
<tr>
<td><strong>IsUnicode</strong> (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><strong>RefCount</strong> (inherited from <strong>TSharedObject</strong>)</td>
<td>Used to return the count of reference to a</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

```pascal
property Compressed: boolean;
```

Remarks
Indicates whether the Blob is compressed. Set this property to True or False to compress or decompress the Blob.

5.12.1.3.2.2 CompressedSize Property

Used to indicate compressed size of the Blob data.

Class
TCompressedBlob

Syntax

```pascal
property CompressedSize: Cardinal;
```

Remarks
Indicates compressed size of the Blob data.

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

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 TSharedObject object.</td>
</tr>
</tbody>
</table>

Methods
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

5.12.1.5.1 Members

TMemData class overview.

© 1997-2021 Devart. All Rights Reserved.  Provide Feedback

5.12.1.6 TObjectType Class

This class is not used.

For a list of all members of this type, see TObjectType members.

© 2021 Devart
Unit

**MemData**

Syntax

```cpp
TObjectType = class(TSharedObject);
```

Inheritance Hierarchy

- **TSharedObject**
- **TObjectType**

© 1997-2021
Devart. All Rights Reserved.

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>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>Increments the reference count for the number of references dependent on the TSharedObject object.</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>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

TObjectType

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.

5.12.1.6.2.4  Size Property

Used to learn the size of an object instance.

Class

TObjectType

Syntax

[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

© 1997-2021
Devart. All Rights Reserved.
5.12.1.6.3 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>
<tr>
<td>FindAttribute</td>
<td>Indicates whether a specified Attribute component is referenced in the TAttributes object.</td>
</tr>
<tr>
<td>Release (inherited from TSharedObject)</td>
<td>Decrements the reference count.</td>
</tr>
</tbody>
</table>

See Also

- TObjectType Class
- TObjectType Class Members

© 1997-2021 Devart. All Rights Reserved.

5.12.1.6.3.1 FindAttribute Method

Indicates whether a specified Attribute component is referenced in the TAttributes object.

Class

TObjectType

Syntax

```delphi
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

**TSharedObject** 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>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>

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

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

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

© 2021 Devart
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

```
procedure AddRef;
```

Remarks

Increments the reference count for the number of references dependent on the TSharedObject object.

See Also

- Release

5.12.1.7.3.2 Release Method

Decrements the reference count.

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>
<tr>
<td>TUpdateRecKinds</td>
<td>Represents the set of TUpdateRecKind.</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.

<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>

5.12.3.1 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 sides. Access time to the field values increases as for cbClient. The time spent on opening DataSet and executing Post decreases. <strong>Note:</strong> 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.</td>
</tr>
<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**

```delphi
TConnLostCause = (clUnknown, clExecute, clOpen, clRefresh, clApply, clServiceQuery, clTransStart, clConnectionApply, clConnect);
```

**Values**
### TDANumericType Enumeration

Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.

#### Unit

**MemData**

#### Syntax

```
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 on client is in TBCD format and is represented as</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

MemData

Syntax

TLocateExOption = (lxCaseInsensitive, lxPartialKey, lxNearest, lxNext, lxUp, lxPartialCompare);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lxCaseInsensitive</td>
<td>Similar to loCaseInsensitive. 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 loPartialKey. 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

MemData

Syntax

TSortType = (stCaseSensitive, stCaseInsensitive, 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>

© 1997-2021 Devart. All Rights Reserved.

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>

© 1997-2021 Devart. All Rights Reserved.
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 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>

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

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>

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback
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

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.

<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</td>
</tr>
</tbody>
</table>
update of rows on database server.

**Prepared**
Determines whether a query is prepared for execution or not.

**Ranged**
Indicates whether a range is applied to a dataset.

**UpdateRecordTypes**
Used to indicate the update status for the current record when cached updates are enabled.

**UpdatesPending**
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>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>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>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</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 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>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>SetRange</strong></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</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></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 ApplyUpdates method while cached updates are</td>
</tr>
</tbody>
</table>
**UpdateStatus**

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</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>
</tbody>
</table>

---

**5.13.1.2 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</td>
</tr>
</tbody>
</table>
### 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`

© 1997-2021 Devart. All Rights Reserved.

### 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**

```
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';
```

---

5.13.1.2.3 KeyExclusive Property

Specifies the upper and lower boundaries for a range.

**Class**

TMemDataSet
### LocalConstraints Property

**Syntax**

```pascal
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 LocalConstrains 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.
5.13.1.1.2.5  LocalUpdate Property

Used to prevent implicit update of rows on database server.

Class

TMemDataSet

Syntax

```object
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.

5.13.1.1.2.6  Prepared Property

Determines whether a query is prepared for execution or not.

Class

TMemDataSet

Syntax

```object
property Prepared: boolean;
```

Remarks

Determines whether a query is prepared for execution or not. The Prepared property currently is not supported by MySQL and is always False.

See Also

- Prepare
5.13.1.2.7 Ranged Property

Indicates whether a range is applied to a dataset.

Class

TMemDataSet

Syntax

property Ranged: Boolean;

Remarks

Use the Ranged property to detect whether a range is applied to a dataset.

See Also

- SetRange
- SetRangeEnd
- SetRangeStart

5.13.1.2.8 UpdateRecordTypes Property

Used to indicate the update status for the current record when cached updates are enabled.

Class

TMemDataSet

Syntax

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

```pascal
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

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>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</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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>SetRange</strong></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</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></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 ApplyUpdates 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>

See Also
- [TMemDataSet Class](#)
- [TMemDataSet Class Members](#)
5.13.1.3.1 ApplyRange Method

Applies a range to the dataset.

Class

TMemDataSet

Syntax

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 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

© 1997-2021 Devart. All Rights Reserved.

5.13.1.3.2 ApplyUpdates Method

Writes dataset's pending cached updates to a database.

Class

TMemDataSet

Overload List
ApplyUpdates

**Description**
Writes dataset's pending cached updates to a database.

**Syntax**

```delphi
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:

```delphi
procedure ApplyButtonClick(Sender: TObject);
begin
  with MyQuery do
```

© 1997-2021 Devart. All Rights Reserved.
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;

See Also
- `TMemDataSet.CachedUpdates`
- `TMemDataSet.CancelUpdates`
- `TMemDataSet.CommitUpdates`
- `TMemDataSet.UpdateStatus`

© 1997-2021 Devart. All Rights Reserved.

Writes dataset's pending cached updates of specified records to a database.

Class
`TMemDataSet`

Syntax

```plaintext
procedure ApplyUpdates(const UpdateRecKinds: TUpdateRecKinds);
overload; virtual;
```

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.

5.13.1.1.3.3 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`
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

```plaintext
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 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
- TMemDataSet.ApplyUpdates
- UpdateStatus

© 1997-2021
Devart. All Rights Reserved.

5.13.1.1.3.6 DeferredPost Method

Makes permanent changes to the database server.

Class
TMemDataSet

Syntax

```
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.

© 1997-2021
Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
5.13.1.1.3.7 EditRangeEnd Method

Enables changing the ending value for an existing range.

Class

TMemDataSet

Syntax

```delphi
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.

5.13.1.1.3.8 EditRangeStart Method

Enables changing the starting value for an existing range.

Class

TMemDataSet

Syntax

```delphi
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

© 1997-2021 Devart. All Rights Reserved.

5.13.1.3.9 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(Field: TField)</td>
<td>Retrieves TBlob object for a field or current record when the field itself is known.</td>
</tr>
<tr>
<td>GetBlob(const FieldName: string)</td>
<td>Retrieves TBlob object for a field or current record when its name is known.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
Retrieves TBlob object for a field or current record when the field itself is known.

**Class**

tMemDataSet

**Syntax**

```vbnet
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.

© 1997-2021 Devart. All Rights Reserved.

Retrieves TBlob object for a field or current record when its name is known.

**Class**

tMemDataSet

**Syntax**

```vbnet
function GetBlob(const FieldName: string): TBlob; overload;
```

**Parameters**

*FieldName*

Holds the name of an existing field.

**Return Value**

TBlob object that was retrieved.

**Example**

MyQuery1.GetBlob('Comment').SaveToFile('Comment.txt');
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><code>Locate(const KeyFields: array of TField; const KeyValues: variant; Options: TLocateOptions)</code></td>
<td>Searches a dataset by the specified fields for a specific record and positions cursor on it.</td>
</tr>
<tr>
<td><code>Locate(const KeyFields: string; const KeyValues: variant; Options: TLocateOptions)</code></td>
<td>Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.</td>
</tr>
</tbody>
</table>

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.

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:

```pascal
with CustTable do
  Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P', '408-431-1000']), [loPartialKey]);
```

See Also

- TMemDataSet.IndexFieldNames
- TMemDataSet.LocateEx

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>LocateEx</td>
<td>Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified fields.</td>
</tr>
<tr>
<td>LocateEx</td>
<td>Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified fields.</td>
</tr>
</tbody>
</table>
TLocateExOptions) method of TDataSet by the specified field names.

Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified fields.

Class

TMemDataSet

Syntax

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.
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.

Provide Feedback
DAC Forum
Request Support

5.13.1.3.12 Prepare Method

Allocates resources and creates field components for a dataset.

Class
TMemDataSet

Syntax

```delphi
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 MySQL prepared protocol has certain server restrictions, and its work is not always
stable. That is why it is advisable to perform test before using preparation in production
versions of applications.

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.

5.13.1.3.13 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
5.13.1.3.14 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.

See Also

- CachedUpdates
- CancelUpdates

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>
</tbody>
</table>
SaveToXML(const FileName: string) Saves the current dataset data to a file in the XML format compatible with ADO format.

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
**TMemDataSet**

**Syntax**

```pascal
procedure SaveToXML(const FileName: string); overload;
```

**Parameters**

*FileName*
- Holds the name of a destination file.

**See Also**
- `TVirtualTable.LoadFromFile`
- `TVirtualTable.LoadFromStream`

---

**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

- ApplyRange
- CancelRange
- EditRangeEnd
- EditRangeStart
- IndexFieldNames
- KeyExclusive
- SetRangeEnd
- SetRangeStart

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
- SetRange
- SetRangeStart

© 1997-2021 Devart. All Rights Reserved.

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.
5.13.1.1.3.19 UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class

TMemDataSet

Syntax

```pascal
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
Class

`TMemDataSet`

Syntax

```delphi
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.

**See Also**

- `CachedUpdates`

© 1997-2021 Devart. All Rights Reserved.

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**

Events 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><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>

See Also
- **TMemDataSet Class**
- **TMemDataSet Class Members**

Occurs when an exception is generated while cached updates are applied to a database.

**Class**

**TMemDataSet**
Syntax

```plaintext
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.NewValue 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](#)

© 1997-2021 Devart. All Rights Reserved.
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.

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 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>

© 1997-2021 Devart. All Rights Reserved.
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 MyDAC 5.20.0.12, 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.

© 1997-2021 Devart. All Rights Reserved.

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 MyDAC 5.20.0.11, 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.
### 5.14 MyAccess

This unit contains implementation of most public classes of MyDAC.

#### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomMyConnection</td>
<td>A base class for connecting to MySQL server.</td>
</tr>
<tr>
<td>TCustomMyConnectionOptions</td>
<td>This class allows setting up the behaviour of the TCustomMyConnection class.</td>
</tr>
<tr>
<td>TCustomMyDataSet</td>
<td>A base class defining functionality for the classes derived from it.</td>
</tr>
<tr>
<td>TCustomMyStoredProc</td>
<td>A class implementing functionality to access stored procedures on a database server.</td>
</tr>
<tr>
<td>TCustomMyTable</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>TMyCommand</td>
<td>A component for execution of SQL statements and stored procedures which do not return rowsets.</td>
</tr>
<tr>
<td>TMyConnection</td>
<td>A component for setting up and controlling connections to MySQL database server.</td>
</tr>
<tr>
<td>TMyConnectionOptions</td>
<td>This class allows setting up the behaviour of the TMyConnection class.</td>
</tr>
<tr>
<td>TMyConnectionSSLOptions</td>
<td>This class allows setting up the behaviour of the TMyConnection class.</td>
</tr>
<tr>
<td>TMyDataSetOptions</td>
<td>This class allows setting up the behaviour of the TMyDataSet class.</td>
</tr>
<tr>
<td>TMyDataSource</td>
<td>TMyDataSource provides an interface between a MyDAC dataset components and data-aware controls on a</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TMyEncryptor</td>
<td>The class that performs encrypting and decrypting of data.</td>
</tr>
<tr>
<td>TMyMetaData</td>
<td>A component for obtaining metainformation about database objects from the server.</td>
</tr>
<tr>
<td>TMyQuery</td>
<td>A component for executing queries and operating record sets. It also provides flexible way to update data.</td>
</tr>
<tr>
<td>TMyStoredProc</td>
<td>A component for accessing and executing stored procedures and functions.</td>
</tr>
<tr>
<td>TMyTable</td>
<td>A component for retrieving and updating data in a single table without writing SQL statements.</td>
</tr>
<tr>
<td>TMyTableOptions</td>
<td>This class allows setting up the behaviour of the TMyTable class.</td>
</tr>
<tr>
<td>TMyTransaction</td>
<td>A component for managing transactions.</td>
</tr>
<tr>
<td>TMyUpdateSQL</td>
<td>A component for tuning update operations for the DataSet component.</td>
</tr>
</tbody>
</table>

**Types**

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyUpdateExecuteEvent</td>
<td>This type is used for the E:Devart.MyDac.TCustomMyDataSet.AfterUpdateExecute and E:Devart.MyDac.TCustomMyDataSet.BeforeUpdateExecute events.</td>
</tr>
</tbody>
</table>

**Enumerations**

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>TLockRecordType</td>
<td>Specifies the type of the</td>
</tr>
</tbody>
</table>
### TLockType

Specifies the type of the table locking.

### TMyIsolationLevel

Specifies the extent to which all outside transactions interfere with subsequent transactions of current connection.

#### Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetServerList</td>
<td>Returns the list of the MySQL servers in LAN. MySQL server does not provide usual ways of such list getting, so it can be incomplete.</td>
</tr>
</tbody>
</table>

#### Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyDACVersion</td>
<td>Read this constant to get current version number for MyDAC.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.  

### 5.14.1 Classes

Classes in the MyAccess unit.

#### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCustomMyConnection</td>
<td>A base class for connecting to MySQL server.</td>
</tr>
<tr>
<td>TCustomMyConnectionOptions</td>
<td>This class allows setting up the behaviour of the TCustomMyConnection class.</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TCustomMyDataSet</td>
<td>A base class defining functionality for the classes derived from it.</td>
</tr>
<tr>
<td>TCustomMyStoredProc</td>
<td>A class implementing functionality to access stored procedures on a database server.</td>
</tr>
<tr>
<td>TCustomMyTable</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>TMyCommand</td>
<td>A component for execution of SQL statements and stored procedures which do not return rowsets.</td>
</tr>
<tr>
<td>TMyConnection</td>
<td>A component for setting up and controlling connections to MySQL database server.</td>
</tr>
<tr>
<td>TMyConnectionOptions</td>
<td>This class allows setting up the behaviour of the TMyConnection class.</td>
</tr>
<tr>
<td>TMyConnectionSSLOptions</td>
<td>This class allows setting up the behaviour of the TMyConnection class.</td>
</tr>
<tr>
<td>TMyDataSetOptions</td>
<td>This class allows setting up the behaviour of the TMyDataSet class.</td>
</tr>
<tr>
<td>TMyDataSource</td>
<td>TMyDataSource provides an interface between a MyDAC dataset components and data-aware controls on a form.</td>
</tr>
<tr>
<td>TMyEncryptor</td>
<td>The class that performs encrypting and decrypting of data.</td>
</tr>
<tr>
<td>TMyMetaData</td>
<td>A component for obtaining metainformation about database objects from the server.</td>
</tr>
<tr>
<td>TMyQuery</td>
<td>A component for executing queries and operating record sets. It also provides flexible way to update data.</td>
</tr>
</tbody>
</table>
**TCustomMyConnection Class**

A base class for connecting to MySQL server.

For a list of all members of this type, see [TCustomMyConnection](#) members.

**Unit**

MyAccess

**Syntax**

```cpp
TCustomMyConnection = class(TCustomDAConnection);
```

**Remarks**

The TCustomMyConnection component is used to establish connection to database server, provide customized login support, and perform transaction control. TCustomMyConnection is the base component for connecting to MySQL server.

**Inheritance Hierarchy**

TCustomDAConnection

TCustomMyConnection

**See Also**
## TCustomMyConnection class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientVersion</td>
<td>Contains the version of the MySQL Client library.</td>
</tr>
<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>Database</td>
<td>Used to specify a database name that is a default source of data for SQL queries once a connection is established.</td>
</tr>
<tr>
<td>InTransaction</td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td>IsolationLevel</td>
<td>Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection.</td>
</tr>
<tr>
<td>LoginPrompt</td>
<td>Specifies whether a login dialog appears immediately before opening a new connection.</td>
</tr>
</tbody>
</table>
### Options

**Options**

Specifies the behaviour of the TMyConnectionOptions object.

**Password** (inherited from **TCustomDAConnection**)

Serves to supply a password for login.

**Pooling** (inherited from **TCustomDAConnection**)

Enables or disables using connection pool.

**PoolingOptions** (inherited from **TCustomDAConnection**)

Specifies the behaviour of connection pool.

**Server** (inherited from **TCustomDAConnection**)

Serves to supply the server name for login.

**ServerVersion**

Holds the version of MySQL server.

**ThreadId**

Used to return the thread ID of the current connection.

**Username** (inherited from **TCustomDAConnection**)

Used to supply a user name for login.

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td><strong>AssignConnect</strong></td>
<td>Shares database connection between the TCustomMyConnection components.</td>
</tr>
<tr>
<td><strong>Commit</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Commits current transaction.</td>
</tr>
<tr>
<td><strong>Connect</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Establishes a connection to the server.</td>
</tr>
<tr>
<td><strong>CreateDataSet</strong></td>
<td>Returns a new instance of TCustomMyDataSet class and associates it with this connection object.</td>
</tr>
<tr>
<td><strong>CreateSQL</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Creates a component for queries execution.</td>
</tr>
<tr>
<td><strong>Disconnect</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Performs disconnect.</td>
</tr>
<tr>
<td><strong>ExecProc</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Allows to execute stored procedure or function providing its name and parameters.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ExecProcEx</td>
<td>Allows to execute a stored procedure or function.</td>
</tr>
<tr>
<td>ExecSQL</td>
<td>Executes any SQL statement outside TMyQuery or TMyCommand components.</td>
</tr>
<tr>
<td>ExecSQLEx</td>
<td>Executes any SQL statement outside the TQuery or TSQl components.</td>
</tr>
<tr>
<td>GetCharsetNames</td>
<td>Populates a string list with the names of available charsets.</td>
</tr>
<tr>
<td>GetDatabaseNames</td>
<td>Returns a database list from the server.</td>
</tr>
<tr>
<td>GetExecuteInfo</td>
<td>Returns the result of the last query execution.</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>GetTriggerNames</td>
<td>Returns a list of triggers from the server.</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>
</tbody>
</table>
RollbackToSavepoint

Cancels all updates for the current transaction.

Savepoint

Defines a point in the transaction to which you can roll back later.

StartTransaction (inherited from TCustomDAConnection)

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>

Properties of the TCustomMyConnection class.

For a complete list of the TCustomMyConnection class members, see the TCustomMyConnection Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientVersion</td>
<td>Contains the version of the MySQL Client library.</td>
</tr>
<tr>
<td>ConnectDialog</td>
<td>Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td>ConnectionTimeout</td>
<td>Used to specify the amount of time to attempt to establish a connection.</td>
</tr>
<tr>
<td>ConnectString</td>
<td>Used to specify the connection information, such as: Username, Password, ...</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ConvertEOL</strong></td>
<td>Allows customizing line breaks in string fields and parameters.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Used to specify a database name that is a default source of data for SQL queries once a connection is established.</td>
</tr>
<tr>
<td><strong>InTransaction</strong></td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td><strong>IsolationLevel</strong></td>
<td>Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection.</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 behaviour of the TMyConnectionOptions object.</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>ServerVersion</strong></td>
<td>Holds the version of MySQL server.</td>
</tr>
<tr>
<td><strong>ThreadId</strong></td>
<td>Used to return the thread ID of the current connection.</td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Used to supply a user name for login.</td>
</tr>
</tbody>
</table>

See Also
- TCustomMyConnection Class
- TCustomMyConnection Class Members
5.14.1.1.2.1  ClientVersion Property

Contains the version of the MySQL Client library.

Class
TCustomMyConnection

Syntax

```pascal
property ClientVersion: string;
```

Remarks

Contains the version of the MySQL Client library (libmysql.dll or libmysqld.dll).

See Also

- TCustomDACConnection.Connect

5.14.1.1.2.2  ConnectionTimeout Property

Used to specify the amount of time to attempt to establish a connection.

Class
TCustomMyConnection

Syntax

```pascal
property ConnectionTimeout: integer default DefValConnectionTimeout;
```

Remarks

Use the ConnectionTimeout property to specify the amount of time to attempt to establish a connection.

Use ConnectionTimeout property to specify the amount of time, in seconds, that can be expired before an attempt to make a connection is considered unsuccessful.
The default value is 15 seconds.

See Also

- TCustomDAConnection.Connect
- TCustomMyDataSet.CommandTimeout
- TMyCommand.CommandTimeout

Class

TCustomMyConnection

Syntax

```
property Database: string;
```

Remarks

Use the Database property to specify a database name that is a default source of data for SQL queries once a connection is established.

Altering Database property makes new database name take effect immediately.

Setting Database='mysql' allows you to omit database specifier in SELECT statements. That is, instead of

```
SELECT * FROM mysql.user;
```

you may just write

```
SELECT * FROM user
```

See Also

- TCustomDAConnection.Server
- TMyConnection.Port
- TCustomDAConnection.Username
- TCustomDAConnection.Password
- TCustomDAConnection.GetDatabaseNames
5.14.1.1.2.4 IsolationLevel Property

Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection.

Class

TCustomMyConnection

Syntax

```pascal
property IsolationLevel: TMyIsolationLevel default ilReadCommitted;
```

Remarks

Use the IsolationLevel property to specify the extent to which all outside transactions interfere with subsequent transactions of current connection.

Changes to IsolationLevel take effect at a time of starting new transaction or opening new connection.

See Also

- TCustomDAConnection.StartTransaction

5.14.1.1.2.5 Options Property

Specifies the behaviour of the TMyConnectionOptions object.

Class

TCustomMyConnection

Syntax

```pascal
property Options: TCustomMyConnectionOptions;
```

Remarks
Set the properties of Options to specify the behaviour of a TMyConnectionOptions 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>Charset</td>
<td>Used to set a character set used by the client.</td>
</tr>
<tr>
<td>NullForZeroDelphiDate</td>
<td>Used to hide the '30-12-1899' dates.</td>
</tr>
<tr>
<td>NumericType</td>
<td>Used to specify the format of storing and representation of the NUMERIC</td>
</tr>
<tr>
<td></td>
<td>(DECIMAL) fields for all TCustomMyDataSet, associated with the given</td>
</tr>
<tr>
<td></td>
<td>connection.</td>
</tr>
<tr>
<td>OptimizedBigInt</td>
<td>Used to convert all fields with field length less than 11 of TLargeIntField</td>
</tr>
<tr>
<td></td>
<td>type into TIntegerField.</td>
</tr>
<tr>
<td>UseUnicode</td>
<td>Used to inform server that all data between client and server sides will</td>
</tr>
<tr>
<td></td>
<td>be passed in Utf8 coding.</td>
</tr>
</tbody>
</table>

See Also
- TCustomDACConnection.Server
- Database
- National Characters

5.14.1.2.6 ServerVersion Property

Holds the version of MySQL server.

Class
TCustomMyConnection

Syntax

```
property ServerVersion: string;
```

Remarks
The version of MySQL server.
See Also
- TCustomDAConnection.Connect

Class
TCustomMyConnection

Syntax

```
property ThreadId: cardinal;
```

Remarks
Use the ThreadId property to return the thread ID of the current connection. This value can be used as an argument to TMyServerControl.KillProcess to kill the thread.

See Also
- TMyServerControl.KillProcess

© 1997-2021
Devar, All Rights Reserved.

5.14.1.1.2.7 ThreadId Property

Used to return the thread ID of the current connection.

Class
TCustomMyConnection

Syntax

```
property ThreadId: cardinal;
```

Remarks
Use the ThreadId property to return the thread ID of the current connection. This value can be used as an argument to TMyServerControl.KillProcess to kill the thread.

See Also
- TMyServerControl.KillProcess

© 1997-2021
Devar, All Rights Reserved.

5.14.1.1.3 Methods

Methods of the TCustomMyConnection class.

For a complete list of the TCustomMyConnection class members, see the TCustomMyConnection Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyUpdates</td>
<td>(inherited from TCustomDACConnection) Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td>AssignConnect</td>
<td>Shares database connection between the</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Commit</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Commits current transaction.</td>
</tr>
<tr>
<td>Connect</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Establishes a connection to the server.</td>
</tr>
<tr>
<td>CreateDataSet</td>
<td>Returns a new instance of TCustomMyDataSet class and associates it with this connection object.</td>
</tr>
<tr>
<td>CreateSQL</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Creates a component for queries execution.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Performs execution.</td>
</tr>
<tr>
<td>ExecProc</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Allows to execute stored procedure or function providing its name and parameters.</td>
</tr>
<tr>
<td>ExecProcEx</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Allows to execute a stored procedure or function.</td>
</tr>
<tr>
<td>ExecSQL</td>
<td></td>
</tr>
<tr>
<td>ExecSQLEx</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Executes any SQL statement outside TQuery or TSQL components.</td>
</tr>
<tr>
<td>GetCharsetNames</td>
<td>Populates a string list with the names of available charsets.</td>
</tr>
<tr>
<td>GetDatabaseNames</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Returns a database list from the server.</td>
</tr>
<tr>
<td>GetExecuteInfo</td>
<td>Returns the result of the last query execution.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Provides a list of available key field names.</td>
</tr>
<tr>
<td>GetStoredProcNames</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Returns a list of stored procedures from the server.</td>
</tr>
<tr>
<td>GetTableNames</td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Provides a list of available tables names.</td>
</tr>
</tbody>
</table>
### GetTriggerNames

Returns a list of triggers from the server.

### MonitorMessage (inherited from TCustomDACConnection)

Sends a specified message through the TCustomDASQLMonitor component.

### Ping (inherited from TCustomDACConnection)

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 TCustomDACConnection)

Marks the connection that should not be returned to the pool after disconnect.

### Rollback (inherited from TCustomDACConnection)

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 roll back later.

### StartTransaction (inherited from TCustomDACConnection)

 Begins a new user transaction.

---

**See Also**

- TCustomMyConnection Class
- TCustomMyConnection Class Members

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.14.1.3.1 AssignConnect Method

Shares database connection between the TCustomMyConnection components.

#### Class

**TCustomMyConnection**

#### Syntax
**procedure** AssignConnect(Source: **TCustomMyConnection**); **virtual**;

**Parameters**

Source
Preconnected TCustomMyConnection component which connection is to be shared with the current TCustomMyConnection component.

**Remarks**

Use the AssignConnect method to share database connection between the TCustomMyConnection components.

AssignConnect assumes that the Source parameter points to a preconnected TCustomMyConnection component which connection is to be shared with the current TCustomMyConnection component. Note that AssignConnect doesn't make any references to the Source TCustomMyConnection component. So before disconnecting parent TCustomMyConnection component call AssignConnect(Nil) or the Disconnect method for all assigned connections.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.1.3.2 CreateDataSet Method

Returns a new instance of TCustomMyDataSet class and associates it with this connection object.

**Class**

**TCustomMyConnection**

**Syntax**

```pascal
function CreateDataSet(AOwner: TComponent = nil): TCustomDADataSet; override;
```

**Return Value**

a new instance of TCustomMyDataSet class.

**Remarks**

The CreateDataSet method returns a new instance of TCustomMyDataSet class and associates it with this connection object.

**See Also**
5.14.1.3.3 ExecSQL Method

Executes any SQL statement outside TMyQuery or TMyCommand components.

Class

TCustomMyConnection

Syntax

```pascal
function ExecSQL(const Text: string; const Params: array of variant): variant; override;
```

Parameters

`Text`

Holds the SQL statement.

`Params`

Holds the array of the parameters values arranged in the same order as they appear in the SQL statement.

Return Value

Null.

Remarks

Call the ExecSQL method to execute any SQL statement outside TMyQuery or TMyCommand components. Supply the Params array with values of parameters arranged in the same order as they appear in SQL statement that is passed in Text string parameter.

**Note:** If a query doesn't have parameters (Params.Count = 0), this query will be executed faster.

© 1997-2021

Devart. All Rights Reserved.

Request Support    DAC Forum    Provide Feedback

5.14.1.3.4 GetCharsetNames Method

Populates a string list with the names of available charsets.

Class
TCustomMyConnection

Syntax

```object
procedure GetCharsetNames(List: TStrings);
```

Parameters

- `List`
  - Holds the string list to populate.

Remarks

Call the GetCharsetName method to populate a string list with the names of available charsets.

**Note:** Any contents already in the target string list object are eliminated and overwritten by the data produces by GetCharsetNames.

See Also

- [Options](#)

Returns the result of the last query execution.

**Class**

TCustomMyConnection

Syntax

```object
function GetExecuteInfo: string;
```

Return Value

- the result of the last query execution.

Remarks

Call the GetExecuteInfo method to returns the result of the last query execution.

The description of the result format you can see at MySQL Reference Manual [mysql_info](#)

**Note:** If you execute a query at [TCustomDADataSet.Execute](#) with
TCustomMyDataSet.FetchAll set to False, a result cannot be retrieved.

Example

The method makes sense for the following SQL statements:

- \texttt{INSERT INTO} ...
- \texttt{SELECT} ...
- \texttt{INSERT INTO} ...
- \texttt{VALUES (\ldots),(\ldots),(\ldots)}
- \texttt{LOAD DATA INFILE} ...
- \texttt{ALTER TABLE}
- \texttt{UPDATE}

See Also

- \texttt{TCustomDADataset.Execute}
- \texttt{TCustomDASQL.Execute}

Returns a list of triggers from the server.

Class

\texttt{TCustomMyConnection}

Syntax

\begin{verbatim}
procedure GetTriggerNames(List: TStrings; AllTriggers: boolean = False);
\end{verbatim}

Parameters

- \textit{List}
  
  A TStrings descendant that will be filled with the names of triggers in the database.

- \textit{AllTriggers}
  
  True, if triggers from all databases are returned. False is only for the current database.

Remarks

Call the \texttt{GetTriggerNames} method to get the names of triggers. \texttt{GetTriggerNames} populates a string list with the names of triggers in the database. If \texttt{AllProcs = True}, the procedure returns to the \textit{List} parameter the names of the triggers that belong to all databases; otherwise, \textit{List} will contain the names of triggers that belong to the current database.
Note: Any contents already in the target string list object are eliminated and overwritten by data produced by GetTriggerNames.

See Also
- TCustomDAConnection.GetDatabaseNames
- TCustomDAConnection.GetTableNames
- TCustomDAConnection.GetStoredProcNames

Releases the specified savepoint without affecting any work that has been performed after its creation.

Class
TCustomMyConnection

Syntax

```pascal
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
5.14.1.3.8 RollbackToSavepoint Method

Cancels all updates for the current transaction.

Class

TCustomMyConnection

Syntax

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

- Savepoint
- TCustomDACConnection.Rollback
- ReleaseSavepoint

© 1997-2021 Devart. All Rights Reserved.

5.14.1.3.9 Savepoint Method

Defines a point in the transaction to which you can roll back later.

Class

TCustomMyConnection

Syntax

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
- `RollbackToSavepoint`
- `ReleaseSavepoint`

5.14.1.2  `TCustomMyConnectionOptions` Class

This class allows setting up the behaviour of the `TCustomMyConnection` class.

For a list of all members of this type, see `TCustomMyConnectionOptions` members.

Unit

`MyAccess`

Syntax

```delphi
TCustomMyConnectionOptions = class(TDAConnectionOptions);
```

Inheritance Hierarchy

`TDAConnectionOptions`

`TCustomMyConnectionOptions`

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AllowImplicitConnect</code> (inherited from <code>TDAConnectionOptions</code>)</td>
<td>Specifies whether to allow or...</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TDACConnectionOptions</strong></td>
<td>not implicit connection opening.</td>
</tr>
<tr>
<td><strong>Charset</strong></td>
<td>Used to set a character set used by the client.</td>
</tr>
<tr>
<td><strong>DefaultSortType</strong> (inherited from TDACConnectionOptions)**</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><strong>DisconnectedMode</strong> (inherited from TDACConnectionOptions)**</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>KeepDesignConnected</strong> (inherited from TDACConnectionOptions)**</td>
<td>Used to prevent an application from establishing a connection at the time of startup.</td>
</tr>
<tr>
<td><strong>LocalFailover</strong> (inherited from TDACConnectionOptions)**</td>
<td>If True, the TCustomDACConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
<tr>
<td><strong>NullForZeroDelphiDate</strong></td>
<td>Used to hide the '30-12-1899' dates.</td>
</tr>
<tr>
<td><strong>NumericType</strong></td>
<td>Used to specify the format of storing and representation of the NUMERIC (DECIMAL) fields for all TCustomMyDataSets, associated with the given connection.</td>
</tr>
<tr>
<td><strong>OptimizedBigInt</strong></td>
<td>Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.</td>
</tr>
<tr>
<td><strong>UseUnicode</strong></td>
<td>Used to inform server that all data between client and server sides will be passed in Utf8 coding.</td>
</tr>
</tbody>
</table>
5.14.1.2.2 Properties

Properties of the `TCustomMyConnectionOptions` class.

For a complete list of the `TCustomMyConnectionOptions` class members, see the `TCustomMyConnectionOptions Members` topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charset</td>
<td>Used to set a character set used by the client.</td>
</tr>
<tr>
<td><strong>DefaultSortType</strong> (inherited from <code>TDACConnectionOptions</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>DisconnectedMode (inherited from <code>TDACConnectionOptions</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>KeepDesignConnected (inherited from <code>TDACConnectionOptions</code>)</td>
<td>Used to prevent an application from establishing a connection at the time of startup.</td>
</tr>
<tr>
<td>LocalFailover (inherited from <code>TDACConnectionOptions</code>)</td>
<td>If True, the <code>TCustomDAConnection.OnConnectionLost</code> event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
<tr>
<td>NullForZeroDelphiDate</td>
<td>Used to hide the '30-12-1899' dates.</td>
</tr>
<tr>
<td><strong>NumericType</strong></td>
<td>Used to specify the format of storing and representation of the NUMERIC (DECIMAL) fields for all</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OptimizedBigInt</td>
<td>Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.</td>
</tr>
<tr>
<td>UseUnicode</td>
<td>Used to inform server that all data between client and server sides will be passed in Utf8 coding.</td>
</tr>
</tbody>
</table>

### Published

#### Name

| AllowImplicitConnect (inherited from TDAConnectionOptions) | Specifies whether to allow or not implicit connection opening. |

### See Also

- TCustomMyConnectionOptions Class
- TCustomMyConnectionOptions Class Members

© 1997-2021 Devart. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback

5.14.1.2.2.1 Charset Property

Used to set a character set used by the client.

### Class

TCustomMyConnectionOptions

### Syntax

```
property charset: string;
```

### Remarks

Use the Charset property to set a character set used by the client. Actually, if this property is enabled, then the "SET NAMES <Charset>" query is executed on establishing a connection. If the UseUnicode property is set, the Charset property will be ignored. The list of available...
character sets you can see by executing the TCustomMyConnection.GetCharsetNames method.

5.14.1.2.2.2 NullForZeroDelphiDate Property

Used to hide the '30-12-1899' dates.

Class

TCustomMyConnectionOptions

Syntax

property NullForZeroDelphiDate: boolean default False;

Remarks

Use the NullForZeroDelphiDate property to hide the '30-12-1899' dates.

If NullForZeroDelphiDate is set to True, the values of all datetime fields will be changed to Null. If the property is set to False, the '30-12-1899' value will be used as an ordinary date. The default value is False.

5.14.1.2.2.3 NumericType Property

Used to specify the format of storing and representation of the NUMERIC (DECIMAL) fields for all TCustomMyDataSet, associated with the given connection.

Class

TCustomMyConnectionOptions

Syntax

property NumericType: T DANumericType default ntFloat;

Remarks

Use the NumericType property to specify the format of storing and representation of the
NUMERIC (DECIMAL) fields for all TCustomMyDataSet, associated with the given connection.

5.14.1.2.2.4 OptimizedBigInt Property

Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.

Class

TCustomMyConnectionOptions

Syntax

```property
OptimizedBigInt: boolean
default False;
```

Remarks

Setting this option converts all fields with field length less than 11 of TLargeIntField type into TIntegerField. This allows to process fields that are results of numeric function or cast values as usual Integer fields. The default value is False.

5.14.1.2.2.5 UseUnicode Property

Used to inform server that all data between client and server sides will be passed in Utf8 coding.

Class

TCustomMyConnectionOptions

Syntax

```property
UseUnicode: boolean
default DefValUseUnicode;
```

Remarks

Informs server that all data between client and server sides will be passed in Utf8 coding.

Setting this option converts all fields of the TStringField type into TWideStringField that allows
to work correctly with symbols of almost all languages simultaneously. On the other hand, it causes a delay in working. If the UseUnicode property is enabled, the CharSet property will be ignored. The default value is False.

5.14.1.3 TCustomMyDataSet Class

A base class defining functionality for the classes derived from it.

For a list of all members of this type, see TCustomMyDataSet members.

Unit

MyAccess

Syntax

TCustomMyDataSet = class(TCustomDADataSet);

Remarks

TCustomMyDataSet is a base dataset component that defines functionality for classes derived from it. Applications never use TCustomMyDataSet objects directly. Instead they use descendants of TCustomMyDataSet, such as TMyQuery and TMyTable that inherit its dataset-related properties and methods.

Inheritance Hierarchy

TMemDataSet
  TCustomDADataSet
    TCustomMyDataSet

See Also

- TMyQuery
- TCustomMyTable
- Master/Detail Relationships
### TCustomMyDataSet 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>CommandTimeout</strong></td>
<td>Used to specify the amount of time to attempt execution of a command.</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></td>
<td>Used to specify a connection object that will be used to connect to a data store.</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>Encryption</strong></td>
<td>Used to specify encryption options in a dataset.</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> (inherited from TCustomDADataset)</td>
<td>Used to define the number of rows to be transferred.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FilterSQL (inherited from TCustomDADataSet)</td>
<td>Used to change the WHERE clause of SELECT statement and reopen a query.</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>InsertId</td>
<td>Returns the ID generated for an AUTO_INCREMENT column by the previous query.</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>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>Specifies when to perform locking of an editing record.</td>
</tr>
<tr>
<td>MacroCount (inherited from TCustomDADataSet)</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros (inherited from TCustomDADataSet)</td>
<td>Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>MasterFields</strong></td>
<td>(inherited from TCustomDADataset) 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 TCustomDADataset) Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Specifies the behaviour of TCustomMyDataSet object.</td>
</tr>
<tr>
<td><strong>ParamCheck</strong></td>
<td>(inherited from TCustomDADataset) 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 TCustomDADataset) Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>(inherited from TCustomDADataset) Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<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>SmartFetch</strong></td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>(inherited from TCustomDADataset) Used to provide a SQL</td>
</tr>
<tr>
<td><strong>Statement</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td>statement that a query component executes when its Open method is called.</td>
<td></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 the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>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><strong>SQLUpdate</strong> (inherited from <strong>TCustomDADataset</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> (inherited from <strong>TCustomDADataset</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 specify an update object component which provides SQL statements that perform updates of the read-only datasets.</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>
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddWhere</strong></td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<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>BreakExec</strong></td>
<td>Breaks execution of the SQL statement on the server.</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>CreateBlobStream</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>DeferredPost</strong></td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong></td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</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>Execute</strong></td>
<td>Overloaded. Executes a SQL statement on the server.</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>TCustomDADataset</td>
</tr>
<tr>
<td>GetBlob</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>GetDataType</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>GetFieldEnum</td>
<td></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>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------</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</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>Overloaded. Locks the current record for the current connection.</td>
</tr>
<tr>
<td>LockTable</td>
<td>Locks table for the current connection.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>OpenNext</td>
<td>Opens next rowset in the statement.</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>RefreshQuick</td>
<td>Retrieves changes posted to the server by another clients on the client side quickly.</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</td>
<td>Marks all records in the cache of updates as</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>UnLockTable</td>
<td></td>
</tr>
<tr>
<td>UnPrepare</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>TMemDataSet</td>
</tr>
</tbody>
</table>
### Properties

Properties of the `TCustomMyDataSet` class.

For a complete list of the `TCustomMyDataSet` class members, see the `TCustomMyDataSet Members` topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UpdateStatus</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>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 insert, delete, update, lock, and refresh operations.</td>
</tr>
<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>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<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>CommandTimeout</strong></td>
<td>Used to specify the amount of time to attempt execution of a command.</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>Used to specify a connection object that will be used to connect to a data store.</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>Encryption</strong></td>
<td>Used to specify encryption options in a dataset.</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> (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 fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>InsertId</strong></td>
<td>Returns the ID generated for an AUTO_INCREMENT column by the previous query.</td>
</tr>
<tr>
<td><strong>IsQuery</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeyFields</strong> (inherited from <strong>TCustomDADataSet</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>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 implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>LockMode</strong></td>
<td>Specifies when to perform locking of an editing record.</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>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</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>Specifies the behaviour of TCustomMyDataSet object.</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>Used to view and set parameter names, values, and data types dynamically.</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, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SmartFetch</td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</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>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>SQLLock</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>SQLRecCount</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>SQLRefresh</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>SQLUpdate</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>UniDirectional</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>UpdateObject</td>
<td></td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>UpdatesPending</td>
<td>TMemDataSet</td>
</tr>
</tbody>
</table>

See Also
- TCustomMyDataSet Class
- TCustomMyDataSet Class Members

© 1997-2021
Devar. All Rights Reserved.  
Request Support  
DAC Forum  
Provide Feedback
5.14.1.3.2.1 CommandTimeout Property

Used to specify the amount of time to attempt execution of a command.

Class

**TCustomMyDataSet**

Syntax

```plaintext
property CommandTimeout: integer default 0;
```

Remarks

Use the CommandTimeout property to specify the amount of time to attempt execution of a command.

Use CommandTimeout to specify the amount of time that expires before an attempt to execute a command is considered unsuccessful. It is measured in seconds.

If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect.

In the case of exceeding waiting time error CR_SERVER_LOST 'Lost connection to MySQL server during query' raises.

The default value is 0 (infinite).

See Also

- **TCustomMyConnection.ConnectionTimeout**
- **TMyCommand.CommandTimeout**

5.14.1.3.2.2 Connection Property

Used to specify a connection object that will be used to connect to a data store.

Class

**TCustomMyDataSet**

Syntax

```plaintext
property Connection: TCustomMyConnection;
```
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 TCustomMyConnection objects.

At runtime, set the Connection property to reference an existing TCustomMyConnection object.

See Also

• TCustomMyConnection

Syntax

```property Encryption: TMyEncryption;
```
property FetchAll: boolean default True;

Remarks

The default value is True.

Note: When setting TCustomMyDataSet.FetchAll = False you should keep in mind that execution of such queries blocks current session. By default MyDAC will create additional session if it is necessary. But this can cause the following problems:

- Each additional session runs outside the transaction context thus TMyConnection.TCustomDACConnection.Commit and TMyConnection.TCustomDACConnection.Rollback operations in main session won't apply changes made in additional sessions.
- Temporary tables created in one session are not accessible from other sessions, therefore simultaneous using of FetchAll = False and temporary tables is impossible.
- TCustomMyDataSet.Lock cannot be used
- LockTable cannot be used

In order to avoid creating additional connection you can set Options to False.

5.14.1.3.2.5 InsertId Property

Returns the ID generated for an AUTO_INCREMENT column by the previous query.

Class

TCustomMyDataSet

Syntax

property InsertId: int64;

Remarks

Use the InsertId property to return the ID generated for an AUTO_INCREMENT column by the previous query. Use this function after you have performed an INSERT query into a table that contains an AUTO_INCREMENT field.

If the query does not perform an insertion into a table that contains an AUTO_INCREMENT field the value of InsertId won't be defined.
InsertId property has sense only if SQL includes INSERT statement within itself. In case of SELECT statements, the value of auto-increment field can be obtained from corresponding table fields.

See Also
- TCustomDADataset.Execute

Class
TCustomMyDataSet

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.

Class
TCustomMyDataSet

Syntax

```property
Options: TMyDataSetOptions;
```

Remarks

Specifies the behaviour of TCustomMyDataSet object.
Set the properties of Options to specify the behaviour of a TCustomMyDataSet 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 TCustomDADataSet.Prepare on a query execution.</td>
</tr>
<tr>
<td>AutoRefresh</td>
<td>Used to automatically refresh dataset every AutoRefreshInterval seconds.</td>
</tr>
<tr>
<td>AutoRefreshInterval</td>
<td>Used to define in what time interval in seconds the Refresh or RefreshQuick method of a Dataset is called.</td>
</tr>
<tr>
<td>BinaryAsString</td>
<td>Used to specify a method of representation of the BINARY and VARBINARY fields.</td>
</tr>
<tr>
<td>CheckRowVersion</td>
<td>Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.</td>
</tr>
<tr>
<td>CreateConnection</td>
<td>Used to specify if an additional connection to a server should be established to execute an additional query in the FetchAll=False mode.</td>
</tr>
<tr>
<td>DefaultValues</td>
<td>Used to fill the DefaultExpression property of TField objects with appropriate value.</td>
</tr>
<tr>
<td>EnableBoolean</td>
<td>Used to specify the method of representation of the TINYINT(1) fields.</td>
</tr>
<tr>
<td>FieldsAsString</td>
<td>Used to store all non-BLOB fields as string (native MySQL format).</td>
</tr>
<tr>
<td>FullRefresh</td>
<td>Used to specify the fields to include in automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. Default value is false.</td>
</tr>
<tr>
<td>NullForZeroDate</td>
<td>Used for MySQL server to represent the value for for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).</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 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 TCustomMyDataSet to quote all field names in autogenerated SQL statements.</td>
</tr>
<tr>
<td><strong>RemoveOnRefresh</strong></td>
<td>Used for dataset to remove a record locally if the RefreshRecord procedure can't find necessary record on the server.</td>
</tr>
<tr>
<td><strong>RequiredFields</strong></td>
<td>Used for TCustomDADataset to set the Required property of TField objects for NOT NULL fields.</td>
</tr>
<tr>
<td><strong>ReturnParams</strong></td>
<td>Used to return the new value of the fields to dataset after insert or update.</td>
</tr>
<tr>
<td><strong>SaveBlobsAsHex</strong></td>
<td>Used to save BLOB data in hexadecimal format in a database after calling the TQuery.SaveToXML method.</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>SetFieldsReadOnly</strong></td>
<td>Used to specify whether fields not belonging to the current updating table get read-only attribute.</td>
</tr>
<tr>
<td><strong>StrictUpdate</strong></td>
<td>Used for TCustomDADataset to raise an exception when the number of updated or deleted records does not equal 1.</td>
</tr>
<tr>
<td><strong>TrimFixedChar</strong></td>
<td>Used to specify whether to discard all trailing spaces in string fields of the dataset.</td>
</tr>
<tr>
<td><strong>TrimVarChar</strong></td>
<td>Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.</td>
</tr>
</tbody>
</table>

5.14.1.3.2.8 SmartFetch Property

The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.

Class

**TCustomMyDataSet**

Syntax

```pascal
property SmartFetch: TSmartFetchOptions;
```
5.14.1.3.2.9 UpdateObject Property

Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.

Class

TCustomMyDataSet

Syntax

```pascal
property UpdateObject: TMyUpdateSQL;
```

Remarks

The UpdateObject property specifies an update object component which provides SQL statements that perform updates of the read-only datasets when cached updates are enabled.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.3.3 Methods

Methods of the TCustomMyDataSet class.

For a complete list of the TCustomMyDataSet class members, see the TCustomMyDataSet 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>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong></td>
<td>Breaks execution of the SQL statement on the server.</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>CreateBlobStream</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>DeferredPost</strong></td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong></td>
<td>Removes WHERE clause from the SQL property and assigns the BaseSQL property.</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>Execute</strong></td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td><strong>Fetched</strong></td>
<td>Used to find out whether TCustomDAD dataSet has fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong></td>
<td>Used to learn whether TCustomDAD dataSet is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong></td>
<td>Used to learn whether</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>FindKey</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindNearest</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) 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>(inherited from <strong>TCustomDADataSet</strong>) Determines if a parameter with the specified name exists in a dataset.</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><strong>GetDataType</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><strong>GetFieldEnum</strong></td>
<td>Retrieve the list of acceptable values for a specified field given by the FieldName parameter.</td>
</tr>
<tr>
<td><strong>GetFieldObject</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Returns a multireference shared object from field.</td>
</tr>
<tr>
<td><strong>GetFieldPrecision</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Retrieves the precision of a number field.</td>
</tr>
<tr>
<td><strong>GetFieldScale</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Retrieves the scale of a number field.</td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Provides a list of available key field names.</td>
</tr>
<tr>
<td><strong>GetOrderBy</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td><strong>GotoCurrent</strong></td>
<td>(inherited from <strong>TCustomDADataSet</strong>) Sets the current record in this dataset similar to the current record in another dataset.</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>Overloaded. Locks the current record for the current connection.</td>
</tr>
<tr>
<td><strong>LockTable</strong></td>
<td>Locks table for the current connection.</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 next rowset in the statement.</td>
</tr>
<tr>
<td><strong>ParamByName</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Sets or uses parameter information for a specific parameter based on its 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>RefreshQuick</strong></td>
<td>Retrieves changes posted to the server by another clients on the client side quickly.</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.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SetOrderBy</strong></td>
<td>(inherited from <code>TCustomDADataSet</code>) Builds an ORDER BY clause of a SELECT statement.</td>
</tr>
<tr>
<td><strong>SetRange</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</strong></td>
<td>(inherited from <code>TMemDataSet</code>) 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 <code>TMemDataSet</code>) 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 <code>TCustomDADataSet</code>) Determines if the SQL property value was saved to the <code>BaseSQL</code> property.</td>
</tr>
<tr>
<td><strong>UnLock</strong></td>
<td>(inherited from <code>TCustomDADataSet</code>) Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnLockTable</strong></td>
<td>Releases a table locked by the <code>TCustomMyDataSet.LockTable</code> method.</td>
</tr>
<tr>
<td><strong>UnPrepare</strong></td>
<td>(inherited from <code>TMemDataSet</code>) 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 <code>TMemDataSet</code>) 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>(inherited from <code>TMemDataSet</code>) Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

**See Also**
- `TCustomMyDataSet Class`
- `TCustomMyDataSet Class Members`
Retrieve the list of acceptable values for a specified field given by the FieldName parameter.

**Class**

`TCustomMyDataSet`

**Syntax**

```pascal
procedure GetFieldEnum(List: TStrings; FieldName: string;
TableName: string = '');
```

**Parameters**

- **List**
  - Holds the list of acceptable values for a specified field.

- **FieldName**
  - Holds the field name.

- **TableName**
  - Holds the table name.

**Remarks**

Call the GetFieldEnum method to retrieve the list of acceptable values for a specified field given by the FieldName parameter. Field should be of the ENUM or the SET type. If GetFieldEnum is called with the TableName parameter empty, TCustomMyDataSet tries to determine table name searching corresponding field name in the fields requested from server.

**Example**

The code presented in Example 1) demonstrates the usage of the GetFieldEnum method. This code can be tested on the sample table presented in Example 2). The result output in memo is shown in Example 3).

```pascal
Example 1)
MyQuery.SQL.Text := 'SELECT `id`, `SET_column` FROM tb_with_set_column';
MyQuery.Open;
MyQuery.GetFieldEnum(Memo.Lines, 'SET_column');

Example 2)
DROP TABLE IF EXISTS tb_enum;
CREATE TABLE `tb_enum` (`uid` INT(11) NOT NULL PRIMARY KEY AUTO_INCREMENT, `c_enum` ENUM('value1','value2','value3') DEFAULT NULL
```
5.14.1.3.3.2  Lock Method

Locks the current record for the current connection.

Class

TCustomMyDataSet

Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lock</strong></td>
<td>Locks the current record for the current connection.</td>
</tr>
<tr>
<td><strong>Lock(LockType: TLockRecordType)</strong></td>
<td>Locks the current record for the current connection.</td>
</tr>
</tbody>
</table>

Remarks

Locks the current record for the current connection. Serves to prevent simultaneous editing of the same record by several users. Makes sense only for InnoDB tables and can be called only inside transaction context.

If any other connection tries to modify the locked record, it will wait for the time specified by `innodb_lock_wait_timeout` server variable, by default 50 seconds. If during this time the record
will not be unlocked, an exception raises.

The record is unlocked on finishing transaction (Commit or Rollback).

**Note:** This method is incompatible with the `TCustomMyDataSet.FetchAll` property set to False.

Lock can be used only with queries returning resultset.

### See Also

- `TCustomDACConnection.StartTransaction`
- `TCustomDACConnection.Commit`
- `TCustomDACConnection.Rollback`
- `TCustomMyDataSet.LockTable`

5.14.1.3.3.3  LockTable Method

Locks the current record for the current connection.

**Class**

`TCustomMyDataSet`

**Syntax**

```plaintext
procedure Lock(LockType: TLockRecordType); reintroduce;
overload;
```

**Parameters**

*LockType*

Holds the type of the record locking.
Syntax

```plaintext
procedure LockTable(LockType: TLockType);
```

Parameters

- **LockType**
  - Specifies the type of the table locking.

Remarks

Call the `LockTable` method to lock table for the current connection. The main purpose of this method is to speed up working with the table.

Table can be released on:
- calling UnlockTable;
- closing connection;
- calling LockTable once more for the same connection.

If a query has several tables then a table specified in `TMyQuery.UpdatingTable` is used.

**Note**: This method is incompatible with the `FetchAll` property set to False.

See Also

- `UnLockTable`
- `TCustomMyDataSet.Lock`

5.14.1.3.3.4 OpenNext Method

Opens next rowset in the statement.

Class

- `TCustomMyDataSet`

Syntax

```plaintext
function OpenNext: boolean;
```

Return Value

- True, if DataSet opens.

Remarks
Call the OpenNext method to get the second and other ResultSets while executing a multiresult query. If DataSet opens, it returns True. If there are no record sets to be represented, it will return False, and the current record set will be closed. Has effect only for the ctDefaultResultSet cursor. The OpenNext method isn't compatible with TCustomDADataSet.Prepare.

Example

```pascal
MyQuery.SQL.Clear;
MyQuery.SQL.Add('SELECT * FROM Table1;');
MyQuery.SQL.Add('SELECT * FROM Table2;');
MyQuery.SQL.Add('SELECT * FROM Table3;');
MyQuery.SQL.Add('SELECT * FROM Table4;');
MyQuery.SQL.Add('SELECT * FROM Table5;');
MyQuery.FetchAll := False;
MyQuery.Open;
repeat
  // < do something >
until not MyQuery.OpenNext;
```

See Also
- TCustomDADataSet.Execute

5.14.1.3.3.5 RefreshQuick Method

Retrieves changes posted to the server by another clients on the client side quickly.

Class
- TCustomMyDataSet

Syntax

```pascal
procedure RefreshQuick(const CheckDeleted: boolean);
```

Parameters

- `CheckDeleted`
  True, if records deleted by another clients will be checked additionally.

Remarks

Call the RefreshQuick method to retrieve changes posted to the server by another clients on the client side quickly. The main difference from the Refresh method is that not all data
corresponding to the query is retrieved on the client, but only the rows which were added or
modified at the moment of the last update. A necessity of data inquiry for each row is defined
by TIMESTAMP field.

If CheckDeleted parameter set to True records deleted by another clients will be checked
additionally.

For RefreshQuick to work it is necessary that a query includes unique key fields and
TIMESTAMP field.

This method is especially effective for queries with huge data level in the single row.

**Note:** If RefreshQuick is called for a dataset which is ordered on the server (query includes
the ORDER BY clause), dataset records ordering can be violated because not all records will
be retrieved by this method. You can use local ordering to solve this problem. For more
information about local ordering, see the [TMemDataSet.IndexFieldNames](#) property
description.

Releases a table locked by the `LockTable` method.

**Class**

`TCustomMyDataSet`

**Syntax**

```pascal
procedure UnLockTable;
```

**Remarks**

Call the UnLockTable method to release a table locked by `LockTable`.

**See Also**

- `LockTable`
- `TCustomMyDataSet.Lock`
5.14.1.4 TCustomMyStoredProc Class

A class implementing functionality to access stored procedures on a database server.

For a list of all members of this type, see `TCustomMyStoredProc` members.

Unit

`MyAccess`

Syntax

```pascal
TCustomMyStoredProc = class(TCustomMyDataSet);
```

Remarks

`TCustomMyStoredProc` implements functionality to access stored procedures on a database server.

You need only to define the `StoredProcName` property, while not bothering about writing a SQL statement manually.

Use the `Execute` method at runtime to generate a request that instructs server to execute procedure and return parameters in the `Params` property.

Stored procedures are supported only for MySQL 5.0.

Inheritance Hierarchy

```
TMemDataSet
  TCustomDADataSet
    TCustomMyDataSet
      TCustomMyStoredProc
```

See Also

- `TCustomMyDataSet`
- `TMyStoredProc`

© 1997-2021 Devart. All Rights Reserved.
## 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>CommandTimeout</strong> (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Used to specify the amount of time to attempt execution of a command.</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>TCustomMyDataSet</strong>)</td>
<td>Used to specify a connection object that will be used to connect to a data store.</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>Encryption</strong> (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>FetchAll</strong> (inherited from <strong>TCustomMyDataSet</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>
</tbody>
</table>
| **FilterSQL** (inherited from **TCustomDADataset**) | Used to change the WHERE
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FinalSQL (inherited from TCustomDADataSet)</td>
<td>clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td>IndexFieldNames (inherited from TMemDataSet)</td>
<td>Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.</td>
</tr>
<tr>
<td>InsertId (inherited from TCustomMyDataSet)</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>Returns the ID generated for an AUTO_INCREMENT column by the previous query.</td>
</tr>
<tr>
<td>KeyExclusive (inherited from TMemDataSet)</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyFields (inherited from TCustomDADataSet)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>LocalConstraints (inherited from TMemDataSet)</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>LocalUpdate (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>LockMode (inherited from TCustomMyDataSet)</td>
<td>Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MacroCount (inherited from TCustomDADataSet)</td>
<td>Specifies when to perform locking of an editing record.</td>
</tr>
<tr>
<td>Macros (inherited from TCustomDADataSet)</td>
<td>Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>MasterFields (inherited from TCustomDADataSet)</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</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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>Specifies the behaviour of TCustomMyDataSet object.</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>Used to view and set parameter names, values, and data types dynamically.</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, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SmartFetch</td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</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>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</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>StoredProcName</td>
<td>Used to specify the name of the stored procedure to call on the server.</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 specify an update object component which provides SQL statements that perform updates of the 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 (inherited from <strong>TCustomDADataset</strong>)</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 (inherited from <strong>TCustomDADataset</strong>)</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 (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>DeferredPost (inherited from <strong>TMemDataSet</strong>)</td>
<td>Makes permanent changes to the database server.</td>
</tr>
<tr>
<td>DeleteWhere (inherited from <strong>TCustomDADataset</strong>)</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>ExecProc</td>
<td>Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Execute (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</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</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>GetFieldEnum</td>
<td>Retrieve the list of acceptable values for a specified field given by the FieldName parameter.</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 server.</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</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>GotoCurrent</td>
<td>(inherited from TCustomDADataSet) Sets the current record in this dataset similar to the current record in another dataset.</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>Lock</td>
<td>(inherited from TCustomMyDataSet) Overloaded. Locks the current record for the current connection.</td>
</tr>
<tr>
<td>LockTable</td>
<td>(inherited from TCustomMyDataSet) Locks table for the current connection.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>(inherited from TCustomDADataSet) Finds a macro with the specified name.</td>
</tr>
<tr>
<td>OpenNext</td>
<td>(inherited from TCustomMyDataSet) Opens next rowset in the statement.</td>
</tr>
<tr>
<td>ParamByName</td>
<td>(inherited from TCustomDADataSet) Sets or uses parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td>Prepare</td>
<td>(inherited from TCustomDADataSet) Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>PrepareSQL</td>
<td>Builds a query for TCustomMyStoredProc based on the Params and StoredProcName properties, and assigns it to the SQL property.</td>
</tr>
<tr>
<td>RefreshQuick</td>
<td>(inherited from TCustomMyDataSet) Retrieves changes posted to the server by another clients on the client side quickly.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-----------------------</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>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>UnLockTable</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>TMemDataSet</td>
</tr>
</tbody>
</table>
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> (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>
5.14.1.4.2 Properties

Properties of the TCustomMyStoredProc class.

For a complete list of the TCustomMyStoredProc class members, see the TCustomMyStoredProc 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>CommandTimeout (inherited from TCustomMyDataSet)</td>
<td>Used to specify the amount of time to attempt execution of a command.</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 TCustomMyDataSet)</td>
<td>Used to specify a connection object that will be used to connect to a data store.</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>Encryption (inherited from TCustomMyDataSet)</td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td>FetchAll (inherited from TCustomMyDataSet)</td>
<td>Defines whether to request all records of the query from database server when the</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</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>InsertId</td>
<td>TCustomMyDataSet</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>LocalConstraints</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LockMode</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>MacroCount</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Macros</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Macros property. 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> (inherited from <strong>TCustomMyDataset</strong>)</td>
<td>Specifies the behaviour of TCustomMyDataSet 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>TCustomDADataset</strong>)</td>
<td>Used to view and set parameter names, values, and data types dynamically.</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>SmartFetch</strong> (inherited from <strong>TCustomMyDataset</strong>)</td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of</td>
</tr>
<tr>
<td><strong>records and to minimize memory consumption.</strong></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</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 the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>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><strong>SQLUpdate</strong> (inherited from <strong>TCustomDADataset</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>StoredProcName</strong></td>
<td>Used to specify the name of the stored procedure to call on the server.</td>
</tr>
<tr>
<td><strong>UniDirectional</strong> (inherited from <strong>TCustomDADataset</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> (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Used to indicate the update status for the current record</td>
</tr>
</tbody>
</table>
when cached updates are enabled.

**UpdatesPending** (inherited from **TMemDataSet**) Used to check the status of the cached updates buffer.

---

### See Also
- **TCustomMyStoredProc Class**
- **TCustomMyStoredProc Class Members**

---

#### 5.14.1.4.2.1 StoredProcName Property

Used to specify the name of the stored procedure to call on the server.

**Class**

**TCustomMyStoredProc**

**Syntax**

```
property StoredProcName: string;
```

**Remarks**

Use the StoredProcName property to specify the name of the stored procedure to call on the server. If StoredProcName does not match the name of an existing stored procedure on the server, then when the application attempts to prepare the procedure prior to execution, an exception is raised.

---

#### 5.14.1.4.3 Methods

Methods of the **TCustomMyStoredProc** class.

For a complete list of the **TCustomMyStoredProc** class members, see the **TCustomMy StoredProc 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>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>EditRangeEnd (inherited from TMemDataSet)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart (inherited from TMemDataSet)</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 (inherited from TCustomDADataSet)</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
</tbody>
</table>
| Executing (inherited from TCustomDADataSet)   | Indicates whether SQL
### TCustomDADataSet Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fetched</strong></td>
<td>Used to find out whether TCustomDADataSet has fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong></td>
<td>Used to learn whether TCustomDADataSet is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong></td>
<td>Used to learn whether TCustomDADataSet is fetching all rows to the end.</td>
</tr>
<tr>
<td><strong>FindKey</strong></td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindNearest</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 KeyValues parameter.</td>
</tr>
<tr>
<td><strong>FindParam</strong></td>
<td>Determines if a parameter with the specified name exists in a dataset.</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>GetDataType</strong></td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><strong>GetFieldEnum</strong></td>
<td>Retrieve the list of acceptable values for a specified field given by the FieldName parameter.</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</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TCustomDADataset</strong></td>
<td>key field names.</td>
</tr>
<tr>
<td>GetOrderBy (inherited from TCustomDADataset)</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GotoCurrent (inherited from TCustomDADataset)</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 (inherited from TCustomMyDataSet)</td>
<td>Overloaded. Locks the current record for the current connection.</td>
</tr>
<tr>
<td>LockTable (inherited from TCustomMyDataSet)</td>
<td>Locks table for the current connection.</td>
</tr>
<tr>
<td>MacroByName (inherited from TCustomDADataset)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>OpenNext (inherited from TCustomMyDataSet)</td>
<td>Opens next rowset in the statement.</td>
</tr>
<tr>
<td>ParamByName (inherited from TCustomDADataset)</td>
<td>Sets or uses parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td>Prepare (inherited from TCustomDADataset)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>PrepareSQL</td>
<td>Builds a query for TCustomMyStoredProc based on the Params and StoredProcName properties, and assigns it to the SQL property.</td>
</tr>
<tr>
<td>RefreshQuick (inherited from TCustomMyDataSet)</td>
<td>Retrieves changes posted to the server by another clients on the client side quickly.</td>
</tr>
<tr>
<td>RefreshRecord (inherited from TCustomDADataset)</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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>
<tr>
<td><strong>SetRange</strong></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</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></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></td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td><strong>UnLock</strong></td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnLockTable</strong></td>
<td>Releases a table locked by the TCustomMyDataSet.LockTable method.</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>
</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.

**See Also**
- **TCustomMyStoredProc Class**
- **TCustomMyStoredProc Class Members**

---

5.14.1.4.3.1 ExecProc Method

Executes a SQL statement on the server.

**Class**

**TCustomMy StoredProc**

**Syntax**

```plaintext
procedure ExecProc;
```

**Remarks**

Call the ExecProc method to execute a SQL statement on the server. If SQL statement is a query, ExecProc calls the Open method.

Internally ExecProc calls inherited **TCustomDADataset.Execute** method and is only included for compatibility with BDE.

**See Also**
- **TCustomDADataset.Execute**

© 1997-2021

Devart. All Rights Reserved.
5.14.1.3.2 PrepareSQL Method

Builds a query for TCustomMyStoredProc based on the Params and StoredProcName properties, and assigns it to the SQL property.

Class
TCustomMyStoredProc

Syntax
procedure PrepareSQL;

Remarks
Call the PrepareSQL method to build a query for TCustomMyStoredProc based on the Params and StoredProcName properties, and assign it to the SQL property. Then the generated query is verified to be valid and, if necessary, the list of parameters is modified.

PrepareSQL is called implicitly when TCustomMyStoredProc is executed.

See Also
- TCustomDADataSet.Params
- StoredProcName
- ExecProc

5.14.1.5 TCustomMyTable 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 TCustomMyTable members.

Unit
MyAccess

Syntax
TCustomMyTable = class(TCustomMyDataSet);

Remarks
TCustomMyTable implements functionality to access data in a table. Use TCustomMyTable properties and methods to gain direct access to records and fields in an underlying server database without writing SQL statements.

Inheritance Hierarchy

TCustomMyDataSet

TCustomMyDataSet

TCustomMyTable

See Also

• TMyTable
• TCustomMyDataSet
• TMyQuery
• Master/Detail Relationships

© 1997-2021
Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.14.1.5.1 Members

TCustomMyTable class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL</td>
<td>(inherited from TCustomDADataset) Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<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>CommandTimeout</td>
<td>(inherited from TCustomMyDataSet) Used to specify the amount of time to attempt execution of a command.</td>
</tr>
<tr>
<td>Conditions</td>
<td>(inherited from TCustomDADataset) Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td>Connection</td>
<td>(inherited from TCustomMyDataSet) 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>DataTypeMap</strong></td>
<td>(inherited from TCustomDADataset) Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>(inherited from TCustomDADataset) 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 TCustomDADataset) 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 TCustomDADataset) Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>Encryption</strong></td>
<td>(inherited from TCustomMyDataSet) Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>FetchAll</strong></td>
<td>(inherited from TCustomMyDataSet) 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 TCustomDADataset) 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 TCustomDADataset) Used to change the WHERE clause of SELECT statement and reopen a query.</td>
</tr>
<tr>
<td><strong>FinalSQL</strong></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><strong>IndexDefs</strong></td>
<td>Contains information about the indexes for a table.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from TMemDataSet) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>InsertId</strong></td>
<td>(inherited from TCustomMyDataSet) Returns the ID generated for an AUTO_INCREMENT column by the previous query.</td>
</tr>
<tr>
<td><strong>IsQuery</strong></td>
<td>(inherited from TCustomDADataset) Used to check whether SQL store.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</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>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>LockMode</td>
<td>Specifies when to perform locking of an editing 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>Offset</td>
<td>Used to allow retrieving data from the server starting from the specified row.</td>
</tr>
<tr>
<td>Options</td>
<td>Specifies the behaviour of the TMyTable object.</td>
</tr>
<tr>
<td>PropName</td>
<td>ParentClass</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>ParamCount</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Params</td>
<td>TCustomDADataSet</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>SmartFetch</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>SQL</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SQLInsert</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>SQLLock</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQLRecCount (inherited from TCustomDADataSet)</td>
<td>to perform a record lock. 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>TableName</td>
<td>Used to specify the name of the database table this component encapsulates.</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 TCustomMyDataSet)</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the 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</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>BreakExec</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>DeleteWhere</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td><strong>TMemDataSet</strong></td>
</tr>
<tr>
<td><strong>EmptyTable</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>Executing</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>Fetched</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td><strong>Fetching</strong></td>
<td><strong>TCustomDADataset</strong></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>FetchingAll</code></td>
<td>Used to learn whether TCustomDADataSet is fetching all rows to the end.</td>
</tr>
<tr>
<td><code>FindKey</code></td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><code>FindMacro</code></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><code>FindNearest</code></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><code>FindParam</code></td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td><code>GetBlob</code></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><code>GetDataType</code></td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><code>GetFieldEnum</code></td>
<td>Retrieve the list of acceptable values for a specified field given by the FieldName parameter.</td>
</tr>
<tr>
<td><code>GetFieldObject</code></td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td><code>GetFieldPrecision</code></td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td><code>GetFieldScale</code></td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td><code>GetKeyFieldNames</code></td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td><code>GetOrderBy</code></td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td><code>GotoCurrent</code></td>
<td>Sets the current record in</td>
</tr>
<tr>
<td>Function</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>TCustomMyDataSet</td>
</tr>
<tr>
<td>LockTable</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>MacroByName</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>OpenNext</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>ParamByName</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Prepare</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>RefreshQuick</td>
<td>TCustomMyDataSet</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>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</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>
<tr>
<td><strong>SetRange</strong></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</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></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></td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td><strong>UnLock</strong></td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td><strong>UnLockTable</strong></td>
<td>Releases a table locked by the TCustomMyDataSet.LockTable method.</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 ApplyUpdates 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>

**Events**

© 2021 Devart
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterExecute</strong></td>
<td>(inherited from TCustomDADataset)Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td><strong>AfterFetch</strong></td>
<td>(inherited from TCustomDADataset)Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td><strong>AfterUpdateExecute</strong></td>
<td>(inherited from TCustomDADataset)Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td><strong>BeforeFetch</strong></td>
<td>(inherited from TCustomDADataset)Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td><strong>BeforeUpdateExecute</strong></td>
<td>(inherited from TCustomDADataset)Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td><strong>OnUpdateError</strong></td>
<td>(inherited from TMemDataSet)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 TMemDataSet)Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.14.1.5.2 Properties

Properties of the TCustomMyTable class.

For a complete list of the TCustomMyTable class members, see the TCustomMyTable 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><strong>CommandTimeout</strong> (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Used to specify the amount of time to attempt execution of a command.</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>TCustomMyDataSet</strong>)</td>
<td>Used to specify a connection object that will be used to connect to a data store.</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>Encryption</strong> (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>FetchAll</strong> (inherited from <strong>TCustomMyDataSet</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>IndexDefs</strong></td>
<td>Contains information about the indexes for a table.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>)&lt;br&gt;Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><strong>InsertId</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>)&lt;br&gt;Returns the ID generated for an AUTO_INCREMENT column by the previous query.</td>
</tr>
<tr>
<td><strong>IsQuery</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>)&lt;br&gt;Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>)&lt;br&gt;Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeyFields</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>)&lt;br&gt;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>Limit</strong></td>
<td></td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>(inherited from <strong>TMemDataSet</strong>)&lt;br&gt;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>)&lt;br&gt;Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>LockMode</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>)&lt;br&gt;Specifies when to perform locking of an editing record.</td>
</tr>
<tr>
<td><strong>MacroCount</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>)&lt;br&gt;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>)&lt;br&gt;Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><strong>MasterFields</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>)&lt;br&gt;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>)&lt;br&gt;Used to specify the data</td>
</tr>
<tr>
<td>Source Component</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Source component which binds current dataset to the master one.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Used to allow retrieving data from the server starting from the specified row.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Options</th>
<th>Specifies the behaviour of the TMyTable object.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ParamCheck (inherited from TCustomDADataset)</th>
<th>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ParamCount (inherited from TCustomDADataset)</th>
<th>Used to indicate how many parameters are there in the Params property.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Params (inherited from TCustomDADataset)</th>
<th>Used to view and set parameter names, values, and data types dynamically.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Prepared (inherited from TMemDataSet)</th>
<th>Determines whether a query is prepared for execution or not.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Ranged (inherited from TMemDataSet)</th>
<th>Indicates whether a range is applied to a dataset.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ReadOnly (inherited from TCustomDADataset)</th>
<th>Used to prevent users from updating, inserting, or deleting data in the dataset.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RefreshOptions (inherited from TCustomDADataset)</th>
<th>Used to indicate when the editing record is refreshed.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RowsAffected (inherited from TCustomDADataset)</th>
<th>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SmartFetch (inherited from TCustomMyDataSet)</th>
<th>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SQL (inherited from TCustomDADataset)</th>
<th>Used to provide a SQL statement that a query component executes when its Open method is called.</th>
</tr>
</thead>
</table>

<p>| SQLDelete (inherited from TCustomDADataset) | Used to specify a SQL statement that will be used |</p>
<table>
<thead>
<tr>
<th><strong>Property</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<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>TableName</strong></td>
<td>Used to specify the name of the database table this component encapsulates.</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 specify an update object component which provides SQL statements that perform updates of the 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**
- TCustomMyTable Class
5.14.1.5.2.1 IndexDefs Property

Contains information about the indexes for a table.

Class

TCustomMyTable

Syntax

property IndexDefs: TIndexDefs;

Remarks

The IndexDefs property is used to contain information about the indexes for a table.

IndexDefs is a collection of index definitions, each of which describes an available index for the table.

In contrast to BDE, can be used only for viewing the list of indexes already created for the table.

As an additional request for the server is required to fill it, IndexDefs is filled on the first call.

5.14.1.5.2.2 Limit Property

Used to set the number of rows retrieved from the query.

Class

TCustomMyTable

Syntax

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

Class
TCustomMyTable

Syntax

```
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
- Limit

Class
TCustomMyTable

Syntax

```
property Options: TMyTableOptions;
```

Remarks
Specifies the behaviour of the TMyTable object.
Set the properties of Options to specify the behaviour of a `TMyTable` 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><code>HandlerIndex</code></td>
<td>Used to assign an index and a value that this index should satisfy.</td>
</tr>
<tr>
<td><code>UseHandler</code></td>
<td>Used for the HANDLER statement to be used instead of the SELECT statement.</td>
</tr>
</tbody>
</table>

5.14.1.5.2.5 `TableName` Property

Used to specify the name of the database table this component encapsulates.

**Class**

`TCustomMyTable`

**Syntax**

```
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.

© 1997-2021 Devart. All Rights Reserved.  

Provide Feedback  

5.14.1.5.3 Methods

Methods of the `TCustomMyTable` class.

For a complete list of the `TCustomMyTable` class members, see the `TCustomMyTable Members` topic.
<table>
<thead>
<tr>
<th>Method</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 data to a BLOB field, specified by the Field parameter.</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>EmptyTable</td>
<td>Deletes all records from the database table specified by the TableName property.</td>
</tr>
<tr>
<td>Execute</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
</tbody>
</table>
| Executing       | Indicates whether SQL
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fetched</strong>&lt;br&gt;(inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td><strong>Fetching</strong>&lt;br&gt;(inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td><strong>FetchingAll</strong>&lt;br&gt;(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>&lt;br&gt;(inherited from <strong>TCustomDADataset</strong>)</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td><strong>FindMacro</strong>&lt;br&gt;(inherited from <strong>TCustomDADataset</strong>)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>FindNearest</strong>&lt;br&gt;(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 KeyValues parameter.</td>
</tr>
<tr>
<td><strong>FindParam</strong>&lt;br&gt;(inherited from <strong>TCustomDADataset</strong>)</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td><strong>GetBlob</strong>&lt;br&gt;(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>&lt;br&gt;(inherited from <strong>TCustomDADataset</strong>)</td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td><strong>GetFieldEnum</strong>&lt;br&gt;(inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Retrieve the list of acceptable values for a specified field given by the FieldName parameter.</td>
</tr>
<tr>
<td><strong>GetFieldObject</strong>&lt;br&gt;(inherited from <strong>TCustomDADataset</strong>)</td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td><strong>GetFieldPrecision</strong>&lt;br&gt;(inherited from <strong>TCustomDADataset</strong>)</td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td><strong>GetFieldScale</strong>&lt;br&gt;(inherited from <strong>TCustomDADataset</strong>)</td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong>&lt;br&gt;(inherited from <strong>TCustomDADataset</strong>)</td>
<td>Provides a list of available</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TCustomDADataSet</strong></td>
<td>key field names.</td>
</tr>
<tr>
<td>GetOrderBy (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
</tr>
<tr>
<td>GotoCurrent (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
</tr>
<tr>
<td>Locate (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>LocateEx (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>Lock (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Overloaded. Locks the current record for the current connection.</td>
</tr>
<tr>
<td>LockTable (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Locks table for the current connection.</td>
</tr>
<tr>
<td>MacroByName (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>OpenNext (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Opens next rowset in the statement.</td>
</tr>
<tr>
<td>ParamByName (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Sets or uses parameter information for a specific parameter based on its name.</td>
</tr>
<tr>
<td>Prepare (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
</tr>
<tr>
<td>RefreshQuick (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Retrieves changes posted to the server by another clients on the client side quickly.</td>
</tr>
<tr>
<td>RefreshRecord (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td>RestoreSQL (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
</tr>
<tr>
<td>RestoreUpdates (inherited from <strong>TMemDataSet</strong>)</td>
<td>Marks all records in the cache of updates as unapplied.</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>UnLockTable</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Releases a table locked by the TCustomMyDataSet.LockTable method.</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>
</tbody>
</table>
**UpdateStatus** (inherited from **TMemDataSet**)  
Indicates the current update status for the dataset when cached updates are enabled.

### See Also
- **TCustomMyTable Class**
- **TCustomMyTable Class Members**

---

**5.14.1.6 TMyCommand Class**

A component for execution of SQL statements and stored procedures which do not return rowsets.

For a list of all members of this type, see **TMyCommand** members.

---

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

---

**5.14.1.5.3.1 EmptyTable Method**

Deletes all records from the database table specified by the TableName property.

#### Class
**TCustomMyTable**

#### Syntax

```plaintext
procedure EmptyTable;
```

#### Remarks

Call the EmptyTable method to delete all records from the database table specified by the TableName property.

#### See Also
- **TableName**
MyAccess

Syntax

TMyCommand = class(TCustomDASQL);

Remarks

Use TMyCommand to access database server using SQL statements.

TMyCommand object in a client application is used mainly to execute SQL statements on
database server. SQL statement should not retrieve records from a database since
TMyCommand does not provide storage for returned data.

Inheritance Hierarchy

TCustomDASQL
   TMyCommand

See Also

• TMyQuery

© 1997-2021
Devart. All Rights
Reserved. Request Support   DAC Forum   Provide Feedback

5.14.1.6.1 Members

TMyCommand 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>Used to specify the amount of time to attempt to execute a command.</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</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FinalSQL</td>
<td>(inherited from TCustomDASQL) Used to return a SQL statement with expanded macros.</td>
</tr>
<tr>
<td>InsertId</td>
<td>Returns the ID generated for an AUTO_INCREMENT column by the previous query.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>(inherited from TCustomDASQL) Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>(inherited from TCustomDASQL) Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>(inherited from TCustomDASQL) 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>(inherited from TCustomDASQL) Indicates the number of parameters in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td>(inherited from TCustomDASQL) Used to contain parameters for a SQL statement.</td>
</tr>
<tr>
<td>ParamValues</td>
<td>(inherited from TCustomDASQL) Used to get or set the values of individual field parameters that are identified by name.</td>
</tr>
<tr>
<td>Prepared</td>
<td>(inherited from TCustomDASQL) Used to indicate whether a query is prepared for execution.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>(inherited from TCustomDASQL) 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>(inherited from TCustomDASQL) Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.</td>
</tr>
</tbody>
</table>

Methods
## MySQL Data Access Components

### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>

### 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>

© 1997-2021 Devart. All Rights Reserved.

5.14.1.6.2 Properties

Properties of the **TMyCommand** class.

For a complete list of the **TMyCommand** class members, see the [TMyCommand 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 with expanded macros.</td>
</tr>
<tr>
<td>InsertId</td>
<td>Returns the ID generated for an AUTO_INCREMENT column by the previous query.</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>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 (inherited from TCustomDASQL)</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>Used to specify the amount of time to attempt to execute a command.</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>
</tbody>
</table>

**See Also**
- TMyCommand Class
- TMyCommand Class Members

---

5.14.1.6.2.1 CommandTimeout Property

Used to specify the amount of time to attempt to execute a command.

**Class**

TMyCommand

**Syntax**

```
property CommandTimeout: integer default 0;
```

**Remarks**

Use the CommandTimeout property to specify the amount of time to attempt to execute a command.

Use CommandTimeout to specify the amount of time that expires before an attempt to
execute a command is considered unsuccessful. Measured in seconds.

If a command executed successfully prior to the expiration of the seconds specified, CommandTimeout has no effect.

In the case of exceeding waiting time error CR_SERVER_LOST 'Lost connection to MySQL server during query' raises.

The default value is 0 (infinite).

See Also
- TCustomMyConnection.ConnectionTimeout
- TCustomMyDataSet.CommandTimeout

© 1997-2021 Devart. All Rights Reserved.
5.14.1.6.2.3 InsertId Property

Returns the ID generated for an AUTO_INCREMENT column by the previous query.

Class

TMyCommand

Syntax

property InsertId: int64;

Remarks

Use the InsertId to return the ID generated for an AUTO_INCREMENT column by the previous query. Use this function after you have performed an INSERT query into a table that contains an AUTO_INCREMENT field.

If the query does not perform an insertion into a table that contains an AUTO_INCREMENT field the value of InsertId won't be defined.

See Also

- TCustomDASQL.Execute

© 1997-2021 Devart. All Rights Reserved.

5.14.1.7 TMyConnection Class

A component for setting up and controlling connections to MySQL database server.

For a list of all members of this type, see TMyConnection members.

Unit

MyAccess

Syntax

TMyConnection = class(TCustomMyConnection);

Remarks

TMyConnection component is used to establish connection to database server, provide customized login support, and perform transaction control. TMyConnection publishes connection-related properties derived from its ancestor class TCustomMyConnection and
introduces specific properties.

Inheritance Hierarchy

TCustomDAConnection
  TCustomMyConnection
  TMyConnection

See Also
- TCustomMyConnection
- TMyEmbConnection

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClientVersion</strong></td>
<td>(inherited from TCustomMyConnection) Contains the version of the MySQL Client library.</td>
</tr>
<tr>
<td><strong>ConnectDialog</strong></td>
<td>(inherited from TCustomDAConnection) Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td><strong>Connected</strong></td>
<td>Used to establish a database connection.</td>
</tr>
<tr>
<td><strong>ConnectionTimeout</strong></td>
<td>(inherited from TCustomMyConnection) Used to specify the amount of time to attempt to establish a connection.</td>
</tr>
<tr>
<td><strong>ConnectString</strong></td>
<td>(inherited from TCustomDAConnection) Used to specify the connection information, such as: UserName, Password, Server, etc.</td>
</tr>
<tr>
<td><strong>ConvertEOL</strong></td>
<td>(inherited from TCustomDAConnection) Allows customizing line breaks in string fields and parameters.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Used to specify the database name.</td>
</tr>
<tr>
<td><strong>HttpOptions</strong></td>
<td>Holds a THttpOptions object that contains settings for</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InTransaction</td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td>IOHandler</td>
<td>Used to assign an external component for communication between MyDAC and MySQL server.</td>
</tr>
<tr>
<td>IsolationLevel</td>
<td>Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection.</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 behaviour of TMyConnection object.</td>
</tr>
<tr>
<td>Password</td>
<td>Used to specify the password.</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>Port</td>
<td>Used to specify the port number for a TCP/IP connection.</td>
</tr>
<tr>
<td>Server</td>
<td>Used to specify the server to connect to.</td>
</tr>
<tr>
<td>ServerVersion</td>
<td>Holds the version of MySQL server.</td>
</tr>
<tr>
<td>SSLOptions</td>
<td>Used to set the properties required for protected SSL connection with the server.</td>
</tr>
<tr>
<td>ThreadId</td>
<td>Used to return the thread ID of the current connection.</td>
</tr>
<tr>
<td>Username</td>
<td>Used to specify the username.</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>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Overloaded. Applies changes in datasets.</td>
</tr>
<tr>
<td><strong>AssignConnect</strong> (inherited from <strong>TCustomMyConnection</strong>)</td>
<td>Shares database connection between the TCustomMyConnection components.</td>
</tr>
<tr>
<td><strong>Commit</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Commits current transaction.</td>
</tr>
<tr>
<td><strong>Connect</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Establishes a connection to the server.</td>
</tr>
<tr>
<td><strong>CreateDataSet</strong> (inherited from <strong>TCustomMyConnection</strong>)</td>
<td>Returns a new instance of TCustomMyDataSet class and associates it with this connection object.</td>
</tr>
<tr>
<td><strong>CreateSQL</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Creates a component for queries execution.</td>
</tr>
<tr>
<td><strong>Disconnect</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Performs disconnect.</td>
</tr>
<tr>
<td><strong>ExecProc</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Allows to execute stored procedure or function providing its name and parameters.</td>
</tr>
<tr>
<td><strong>ExecProcEx</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Allows to execute a stored procedure or function.</td>
</tr>
<tr>
<td><strong>ExecSQL</strong> (inherited from <strong>TCustomMyConnection</strong>)</td>
<td>Executes any SQL statement outside TMyQuery or TMyCommand components.</td>
</tr>
<tr>
<td><strong>ExecSQLEx</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Executes any SQL statement outside the TQuery or TSQI components.</td>
</tr>
<tr>
<td><strong>GetCharsetNames</strong> (inherited from <strong>TCustomMyConnection</strong>)</td>
<td>Populates a string list with the names of available charsets.</td>
</tr>
<tr>
<td><strong>GetDatabaseNames</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Returns a database list from the server.</td>
</tr>
<tr>
<td><strong>GetExecuteInfo</strong> (inherited from <strong>TCustomMyConnection</strong>)</td>
<td>Returns the result of the last query execution.</td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td><strong>GetStoredProcNames</strong> (inherited from <strong>TCustomDAConnection</strong>)</td>
<td>Returns a list of stored procedures from the server.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetTableNames</td>
<td>Provides a list of available tables names.</td>
</tr>
<tr>
<td>GetTriggerNames</td>
<td>Returns a list of triggers from the server.</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>Begins a new user transaction.</td>
</tr>
</tbody>
</table>

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>
5.14.1.7.2 Properties

Properties of the TMyConnection class.

For a complete list of the TMyConnection class members, see the TMyConnection Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientVersion</td>
<td>Contains the version of the MySQL Client library.</td>
</tr>
<tr>
<td>ConnectDialog</td>
<td>Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td>ConnectionTimeout</td>
<td>Used to specify the amount of time to attempt to establish a connection.</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>IsolationLevel</td>
<td>Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection.</td>
</tr>
<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>PoolingOptions</td>
<td>Specifies the behaviour of connection pool.</td>
</tr>
<tr>
<td>ServerVersion</td>
<td>Holds the version of MySQL server.</td>
</tr>
<tr>
<td>ThreadId</td>
<td>Used to return the thread ID of the current connection.</td>
</tr>
</tbody>
</table>
### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected</td>
<td>Used to establish a database connection.</td>
</tr>
<tr>
<td>Database</td>
<td>Used to specify the database name.</td>
</tr>
<tr>
<td>HttpOptions</td>
<td>Holds a THttpOptions object that contains settings for HTTP connection.</td>
</tr>
<tr>
<td>IOHandler</td>
<td>Used to assign an external component for communication between MyDAC and MySQL server.</td>
</tr>
<tr>
<td>Options</td>
<td>Specifies the behaviour of TMyConnection object.</td>
</tr>
<tr>
<td>Password</td>
<td>Used to specify the password.</td>
</tr>
<tr>
<td>Port</td>
<td>Used to specify the port number for a TCP/IP connection.</td>
</tr>
<tr>
<td>Server</td>
<td>Used to specify the server to connect to.</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>Used to specify the username.</td>
</tr>
</tbody>
</table>

### See Also
- [TMyConnection Class](#)
- [TMyConnection Class Members](#)

5.14.1.7.2.1 Connected Property

**Connected Property**

Used to establish a database connection.

### Class

[TMyConnection](#)
**Syntax**

```javascript
property Connected stored IsConnectedStored;
```

**Remarks**

Indicates whether the database connection is active. Set this property to True to establish a database connection. Setting this property to False will close a connection.

**Class**

**TMyConnection**

**Syntax**

```javascript
property Database: string;
```

**Remarks**

Use the Database property to specify the name of a database to be used as the default source of data for SQL queries after establishing a connection.

Changing the value of this property takes effect immediately.

You can use the Database property to omit the database name in SELECT statements. For example, you can write

```sql
SELECT * FROM user;
```

instead of

```sql
SELECT * FROM mysql.user;
```

**See Also**

- Server
- Port
- Username
- Password
- **TCustomDAConnection.GetDatabaseNames**
5.14.1.7.2.3  HttpOptions Property

Holds a THttpOptions object that contains settings for HTTP connection.

Class

TMyConnection

Syntax

property HttpOptions: THttpOptions;

Remarks

The HttpOptions property holds a THttpOptions object that contains settings for HTTP connection.

For more information on HTTP tunneling refer to the Network Tunneling article.

5.14.1.7.2.4  IOHandler Property

Used to assign an external component for communication between MyDAC and MySQL server.

Class

TMyConnection

Syntax

property IOHandler: TCRIOHandler;

Remarks

Use the IOHandler property to assign an external component for communication between MyDAC and MySQL server. The component must be a descendant of the TMyIOHandler abstract class.

There is an example of implementation and usage of such component in the SecureBridge
demo. This component provides integration with the SecureBridge library, so SecureBridge should be also installed to build and install it. You can read more about this demo in the Demo Projects topic, and in the Readme.html file located in the the SecureBridge demo directory.

SecureBridge is a library that can be used for ensuring protection of important data transferred between MySQL server and MyDAC based applications through public networks.

See Also
• TMyIOHandler

Specifications of the behaviour of TMyConnection object.

Class
TMyConnection

Syntax

```
property Options: TMyConnectionOptions;
```

Remarks
Set the properties of Options to specify the behaviour of a TMyConnection 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>Charset</td>
<td>Used to set the character set that MyDAC uses to read and write character data.</td>
</tr>
<tr>
<td>CheckBackslashes</td>
<td>Used to check the value of the NO_BACKSLASH_ESCAPES server variable.</td>
</tr>
<tr>
<td>Compress</td>
<td>Used to apply compression on transferring data.</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><strong>Direct</strong></td>
<td>Used to work without using MySQL client library (libmysql.dll).</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DisconnectedMode</strong></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>Embedded</strong></td>
<td>Used to specify the server that will be used to connect.</td>
</tr>
<tr>
<td><strong>Interactive</strong></td>
<td>Permit interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable is set to the value of the session interactive_timeout variable.</td>
</tr>
<tr>
<td><strong>IPVersion</strong></td>
<td>Used to specify Internet Protocol Version.</td>
</tr>
<tr>
<td><strong>KeepDesignConnected</strong></td>
<td>Used to prevent an application from establishing a connection at startup.</td>
</tr>
<tr>
<td><strong>LocalFailover</strong></td>
<td>If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.</td>
</tr>
<tr>
<td><strong>NullForZeroDelphiDate</strong></td>
<td>Used to hide the '30-12-1899' dates.</td>
</tr>
<tr>
<td><strong>NumericType</strong></td>
<td>Specifies the format of storing and representing NUMERIC (DECIMAL) fields.</td>
</tr>
<tr>
<td><strong>OptimizedBigInt</strong></td>
<td>Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.</td>
</tr>
<tr>
<td><strong>Protocol</strong></td>
<td>Used to specify the protocol to use when connecting to server.</td>
</tr>
<tr>
<td><strong>UseUnicode</strong></td>
<td>Used to specify whether the UTF8 charset will be used.</td>
</tr>
</tbody>
</table>

**See Also**
- TCustomDAConnection.Server
- TCustomMyConnection.Database
- TCustomMyConnection.Options
- Embedded Server

© 1997-2021 Devart. All Rights Reserved.
5.14.1.7.2.6 Password Property

Used to specify the password.

Class

TMyConnection

Syntax

```
property Password: string;
```

Remarks

Use the Password property to specify the password of your MySQL account. When the value of the property is changed, TMyConnection calls the Disconnect method.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.7.2.7 Port Property

Used to specify the port number for a TCP/IP connection.

Class

TMyConnection

Syntax

```
property Port: integer default MYSQL_PORT;
```

Remarks

Use the Port property to specify the port number for a TCP/IP connection. Note that TCustomDACConnection.Server property determines the type of the connection.

The default value is 3306.

The Port property can be used only if TCustomMyConnection.Options is set to False.

See Also

- TCustomDACConnection.Server
- TCustomMyConnection.Database
5.14.1.7.2.8 Server Property

Used to specify the server to connect to.

Class

TMyConnection

Syntax

```text
property Server: string;
```

Remarks

Use the Server property to specify the server name. When the value of the property is changed, TMyConnection calls the Disconnect method.

5.14.1.7.2.9 SSLOptions Property

Used to set the properties required for protected SSL connection with the server.

Class

TMyConnection

Syntax

```text
property SSLOptions: TMyConnectionSSLOptions;
```

Remarks

Use the SSLOptions property to set the properties required for protected SSL connection with the server. These properties can be used only for Options.Protocol = mpSSL.

For using ssleay32.dll and libeay32.dll files are needed.

The detailed description of these properties you can find in MySQL Reference Manual

See Also

- Options
5.14.1.7.2.10 Username Property

Used to specify the username.

Class

TMyConnection

Syntax

```
property Username: string;
```

Remarks

Use the Password property to specify the username of your MySQL account.

When the value of the property is changed, TMyConnection calls the Disconnect method.

5.14.1.8 TMyConnectionOptions Class

This class allows setting up the behaviour of the TMyConnection class.

For a list of all members of this type, see TMyConnectionOptions members.

Unit

MyAccess

Syntax

```
TMyConnectionOptions = class(TCustomMyConnectionOptions);
```

Inheritance Hierarchy

```
TDACConnectionOptions
  TCustomMyConnectionOptions
  TMyConnectionOptions
```

© 1997-2021 Devart. All Rights Reserved.
5.14.1.8.1 Members

**TMyConnectionOptions** 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><strong>Charset</strong></td>
<td>Used to set the character set that MyDAC uses to read and write character data.</td>
</tr>
<tr>
<td>CheckBackslashes</td>
<td>Used to check the value of the NO_BACKSLASH_ESCAPES server variable.</td>
</tr>
<tr>
<td>Compress</td>
<td>Used to apply compression on transferring data.</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 <strong>TMemDataSet.IndexFieldNames</strong> property of a dataset.</td>
</tr>
<tr>
<td>Direct</td>
<td>Used to work without using MySQL client library (libmysql.dll).</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>Embedded</td>
<td>Used to specify the server that will be used to connect.</td>
</tr>
<tr>
<td>Interactive</td>
<td>Permit interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable is set to the value of the session.</td>
</tr>
</tbody>
</table>
### Properties of the TMyConnectionOptions class

For a complete list of the TMyConnectionOptions class members, see the TMyConnectionOptions Members topic.

#### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowImplicitConnect (inherited from TDAConnectionOptions)</td>
<td>Specifies whether to allow or not implicit connection opening.</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Charset</td>
<td>Used to set the character set that MyDAC uses to read and write character data.</td>
</tr>
<tr>
<td>CheckBackslashes</td>
<td>Used to check the value of the NO_BACKSLASH_ESCAPES server variable.</td>
</tr>
<tr>
<td>Compress</td>
<td>Used to apply compression on transferring data.</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>Direct</td>
<td>Used to work without using MySQL client library (libmysql.dll).</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>Embedded</td>
<td>Used to specify the server that will be used to connect.</td>
</tr>
<tr>
<td>Interactive</td>
<td>Permit interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable is set to the value of the session interactive_timeout variable.</td>
</tr>
<tr>
<td>IPVersion</td>
<td>Used to specify Internet Protocol Version.</td>
</tr>
<tr>
<td>KeepDesignConnected</td>
<td>Used to prevent an application from establishing a connection at startup.</td>
</tr>
<tr>
<td>LocalFailover</td>
<td>If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NullForZeroDelphiDate</td>
<td>Used to hide the '30-12-1899' dates.</td>
</tr>
<tr>
<td>NumericType</td>
<td>Specifies the format of storing and representing NUMERIC (DECIMAL) fields.</td>
</tr>
<tr>
<td>OptimizedBigInt</td>
<td>Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Used to specify the protocol to use when connecting to server.</td>
</tr>
<tr>
<td>UseUnicode</td>
<td>Used to specify whether the UTF8 charset will be used.</td>
</tr>
</tbody>
</table>

**See Also**
- [TMyConnectionOptions Class](#)
- [TMyConnectionOptions Class Members](#)

**Charset Property**

Used to set the character set that MyDAC uses to read and write character data.

**Class**

TMyConnectionOptions

**Syntax**

```property
charset: string;
```

**Remarks**

Use the Charset property to set the character set that MyDAC uses to read and write character data.
### 5.14.1.8.2.2 CheckBackslashes Property

Used to check the value of the NO_BACKSLASH_ESCAPES server variable.

**Class**

*TMyConnectionOptions*

**Syntax**

```plaintext
property CheckBackslashes: boolean default False;
```

**Remarks**

Use the CheckBackslashes property to check the value of the NO_BACKSLASH_ESCAPES server variable. It enables or disables the usage of the backslash character (\) as an escape character within strings. The backslash character is used when transferring parameters in a query and when dumping a database to a SQL script with the TMyDump component.

If the CheckBackslashes property is set to True, the value of the NO_BACKSLASH_ESCAPES variable is read from the server when establishing a connection. The received value will determine if the backslash character is used as an escape character within strings.

If the CheckBackslashes property is set to False, the backslash character will be used in any case. The default value is False.

### 5.14.1.8.2.3 Compress Property

Used to apply compression on transferring data.

**Class**

*TMyConnectionOptions*

**Syntax**

```plaintext
property Compress: boolean default DefValCompress;
```

**Remarks**

Use the Compress property to use compression on transferring data. Setting this property to
True is quite effective on transferring big volumes of data through slow connection. Pay attention that each row is compressed separately. Be careful when setting this option as in some cases it may decrease fetch speed instead of increasing. This property is ignored under CLR. The default value is False.

5.14.1.8.2.4 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

TMyConnectionOptions

Syntax

```
property DefaultSortType: TSortType;
```

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.

5.14.1.8.2.5 Direct Property

Used to work without using MySQL client library (libmysql.dll).

Class

TMyConnectionOptions

Syntax

```
property Direct: boolean default DefValDirect;
```

Remarks
Use the Direct property to work without using MySQL client library (libmysql.dll). Used only if Embedded is disabled.

5.14.1.8.2.6 DisconnectedMode Property

Used to open a connection only when needed for performing a server call and closes after performing the operation.

Class

TMyConnectionOptions

Syntax

```plaintext
property DisconnectedMode: boolean;
```

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.

5.14.1.8.2.7 Embedded Property

Used to specify the server that will be used to connect.

Class

TMyConnectionOptions

Syntax

```plaintext
property Embedded: boolean default DefValEmbedded;
```

Remarks
Use the Embedded property to specify what server will be used to connect - MySQL server or Embedded MySQL server. You can read about the features and using Embedded server at the Embedded Server. In most cases, you should use TMyEmbConnection instead of this option.

### Interactive Property

Permit interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable is set to the value of the session interactive_timeout variable.

**Class**

TMyConnectionOptions

**Syntax**

```property Interactive: boolean default DefValInteractive;
```

### IPVersion Property

Used to specify Internet Protocol Version.

**Class**

TMyConnectionOptions

**Syntax**

```property IPVersion: TIPVersion default DefValIPVersion;
```

**Remarks**

Use the IPVersion property to specify Internet Protocol Version.

Supported values:

- `ivIPBoth` - specifies that either Internet Protocol Version 6 (IPv6) or Version 4 (IPv4) will be used;
- IPv4 (default) - specifies that Internet Protocol Version 4 (IPv4) will be used;
- IPv6 - specifies that Internet Protocol Version 6 (IPv6) will be used.

**Note:** When the TIPVersion property is set to IPv6Both, a connection attempt will be made via IPv6 if it is enabled on the operating system. If the connection attempt fails, a new connection attempt will be made via IPv4.

**See Also**
- TIPVersion

5.14.1.8.2.10 KeepDesignConnected Property

Used to prevent an application from establishing a connection at startup.

**Class**

**TMySQLConnectionOptions**

**Syntax**

```property
KeepDesignConnected: boolean;
```

**Remarks**

The KeepDesignConnected property prevents application from establishing a connection at startup 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.

5.14.1.8.2.11 LocalFailover Property

If True, the **TCustomDAConnection.OnConnectionLost** event occurs and a failover operation can be performed after connection breaks.

**Class**

**TMySQLConnectionOptions**
**LocalFailover Property**

```delphi
property LocalFailover: boolean;
```

**Remarks**

If True, the `TCustomDACConnection.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.

---

**NullForZeroDelphiDate Property**

Used to hide the ‘30-12-1899’ dates.

**Class**

`TMyConnectionOptions`

**Syntax**

```delphi
property NullForZeroDelphiDate: boolean default False;
```

**Remarks**

Use the `NullForZeroDelphiDate` property to hide the ‘30-12-1899’ dates. If `NullForZeroDelphiDate` is set to True, the values of all datetime fields will be changed to Null. If the property is set to False, the ‘30-12-1899’ value will be used as an ordinary date. The default value is False.

---

**NumericType Property**

Specifies the format of storing and representing NUMERIC (DECIMAL) fields.

**Class**

`TMyConnectionOptions`

**Syntax**

```delphi
```

© 2021 Devart

© 1997-2021 Devart. All Rights Reserved.
**property** NumericType: *TDANumericType default ntFloat;*

### Remarks

Use the NumericType property to specify the format of storing and representing NUMERIC (DECIMAL) fields.

- **ntBCD** - Data is stored on the client side as currency and represented as TBCDField. This format allows storing data with precision up to 0.0001.
- **ntFloat** - Data stored on the client side is in double format and represented as TFloatField. The default value.
- **ntFmtBCD** - Data is represented as TFMTBCDField. TFMTBCDField gives greater precision and accuracy than TBCDField, but is slower.

---

5.14.1.8.2.14 OptimizedBigInt Property

Used to convert all fields with field length less than 11 of TLargeIntField type into TIntegerField.

### Class

*TMyConnectionOptions*

### Syntax

**property** OptimizedBigInt: boolean *default* False;

### Remarks

Setting this option converts all fields with field length less than 11 of TLargeIntField type into TIntegerField. This allows to process fields that are results of numeric function or cast values as usual Integer fields. The default value is False.

---

5.14.1.8.2.15 Protocol Property

Used to specify the protocol to use when connecting to server.

### Class
TMyConnectionOptions

Syntax


Remarks

Use the Protocol property to specify which protocol to use when connecting to server. To use these constants you should add MyClasses unit to uses clause.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.8.2.16 UseUnicode Property

Used to specify whether the UTF8 charset will be used.

Class

TMyConnectionOptions

Syntax

property UseUnicode: boolean default DefValUseUnicode;

Remarks

Use the UseUnicode property to set MySQL client charset to UTF8 and convert client data according to this charset. When a value is assigned to this property, TMyConnection is closed.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.9 TMyConnectionSSLOptions Class

This class allows setting up the behaviour of the TMyConnection class.

For a list of all members of this type, see TMyConnectionSSLOptions members.

Unit

MyAccess
**Syntax**

```objectivec
TMyConnectionSSLOptions = class(TDAConnectionSSLOptions);
```

**Inheritance Hierarchy**

- `TDAConnectionSSLOptions`
  - `TMyConnectionSSLOptions`

© 1997-2021 Devart. All Rights Reserved.

### TMyConnectionSSLOptions Class Overview

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACert</td>
<td>(inherited from <code>TDAConnectionSSLOptions</code>) Holds the path to the certificate authority file.</td>
</tr>
<tr>
<td>Cert</td>
<td>(inherited from <code>TDAConnectionSSLOptions</code>) Holds the path to the client certificate.</td>
</tr>
<tr>
<td>CipherList</td>
<td>(inherited from <code>TDAConnectionSSLOptions</code>) Holds the list of allowed SSL ciphers.</td>
</tr>
<tr>
<td>Key</td>
<td>(inherited from <code>TDAConnectionSSLOptions</code>) Holds the path to the private client key.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

### TMyDataSetOptions Class

This class allows setting up the behaviour of the TMyDataSet class.

For a list of all members of this type, see `TMyDataSetOptions` members.

**Unit**

`MyAccess`

**Syntax**

```objectivec
TMyDataSetOptions = class(TDADatasetOptions);
```
Inheritance Hierarchy

**TDADatasetOptions**
**TMyDataSetOptions**

© 1997-2021 Devart. All Rights Reserved.

5.14.1.10.1 Members

**TMyDataSetOptions** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPrepare</td>
<td>Used to execute automatic TCustomDADDataSet.Prepare on a query execution.</td>
</tr>
<tr>
<td>AutoRefresh</td>
<td>Used to automatically refresh dataset every AutoRefreshInterval seconds.</td>
</tr>
<tr>
<td>AutoRefreshInterval</td>
<td>Used to define in what time interval in seconds the Refresh or TCustomMyDataSet.RefreshQuick method of a DataSet is called.</td>
</tr>
<tr>
<td>BinaryAsString</td>
<td>Used to specify a method of representation of the BINARY and VARBINARY fields.</td>
</tr>
<tr>
<td>CacheCalcFields</td>
<td>(inherited from <strong>TDADatasetOptions</strong>) Used to enable caching of the TField.Calculated and TField.Lookup fields.</td>
</tr>
<tr>
<td>CheckRowVersion</td>
<td>Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.</td>
</tr>
<tr>
<td>CompressBlobMode</td>
<td>(inherited from <strong>TDADatasetOptions</strong>) Used to store values of the BLOB fields in compressed state.</td>
</tr>
<tr>
<td><strong>TDADatasetOptions</strong></td>
<td>form.</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>CreateConnection</strong></td>
<td>Used to specify if an additional connection to a server should be established to execute an additional query in the TCustomMyDataSet.FetchAll=False mode.</td>
</tr>
<tr>
<td><strong>DefaultValues</strong></td>
<td>Used to fill the DefaultExpression property of TField objects with appropriate value.</td>
</tr>
<tr>
<td><strong>DetailDelay</strong> (inherited from <strong>TDADatasetOptions</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>EnableBoolean</strong></td>
<td>Used to specify the method of representation of the TINYINT(1) fields.</td>
</tr>
<tr>
<td><strong>FieldsAsString</strong></td>
<td>Used to store all non-BLOB fields as string (native MySQL format).</td>
</tr>
<tr>
<td><strong>FlatBuffers</strong> (inherited from <strong>TDADatasetOptions</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 automatically generated SQL statement when calling the TCustomDADataset.RefreshRecord method. Default value is false.</td>
</tr>
<tr>
<td><strong>InsertAllSetFields</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to include all set dataset fields in the generated INSERT statement.</td>
</tr>
<tr>
<td><strong>LocalMasterDetail</strong> (inherited from <strong>TDADatasetOptions</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>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LongStrings</strong> (inherited from <strong>TDADDataSetOptions</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>TDADDataSetOptions</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>NullForZeroDate</strong></td>
<td>Used for MySQL server to represent the value for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).</td>
</tr>
<tr>
<td><strong>NumberRange</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></td>
<td>Used for TCustomDADDataSet to perform additional query to get 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 TCustomMyDataSet to quote all field names in autogenerated SQL statements.</td>
</tr>
<tr>
<td><strong>RemoveOnRefresh</strong></td>
<td>Used for dataset to remove a record locally if the RefreshRecord procedure can't find necessary record on the server.</td>
</tr>
<tr>
<td><strong>RequiredFields</strong></td>
<td>Used for TCustomDADDataSet to set the Required property of TField objects for NOT NULL fields.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ReturnParams</strong></td>
<td>Used to return the new value of the fields to dataset after insert or update.</td>
</tr>
<tr>
<td><strong>SaveBlobsAsHex</strong></td>
<td>Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML method.</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>SetFieldsReadOnly</strong></td>
<td>Used to specify whether fields not belonging to the current updating table get read-only attribute.</td>
</tr>
<tr>
<td><strong>StrictUpdate</strong></td>
<td>Used for TCustomDADataset to raise an exception when the number of updated or deleted records does not equal 1.</td>
</tr>
<tr>
<td><strong>TrimFixedChar</strong></td>
<td>Used to specify whether to discard all trailing spaces in string fields of the dataset.</td>
</tr>
<tr>
<td><strong>TrimVarChar</strong></td>
<td>Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.</td>
</tr>
<tr>
<td><strong>UpdateAllFields</strong></td>
<td>(inherited from <strong>TDADatasetOptions</strong>) Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
<tr>
<td><strong>UpdateBatchSize</strong></td>
<td>(inherited from <strong>TDADatasetOptions</strong>) 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>
5.14.1.10.2 Properties

Properties of the **TMyDataSetOptions** class.

For a complete list of the **TMyDataSetOptions** class members, see the **TMyDataSetOptions Members** topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>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>
</tbody>
</table>
### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MasterFieldsNullable</strong></td>
<td>(inherited from <strong>TDADataSetOptions</strong>) 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>UpdateAllFields</strong></td>
<td>(inherited from <strong>TDADataSetOptions</strong>) Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
<tr>
<td><strong>UpdateBatchSize</strong></td>
<td>(inherited from <strong>TDADataSetOptions</strong>) 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>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoPrepare</strong></td>
<td>Used to execute automatic <code>TCustomDADataSet.Prepare</code> on a query execution.</td>
</tr>
<tr>
<td><strong>AutoRefresh</strong></td>
<td>Used to automatically refresh dataset every <code>AutoRefreshInterval</code> seconds.</td>
</tr>
<tr>
<td><strong>AutoRefreshInterval</strong></td>
<td>Used to define in what time interval in seconds the <code>Refresh</code> or <code>TCustomMyDataSet.RefreshQuick</code> method of a DataSet is called.</td>
</tr>
<tr>
<td><strong>BinaryAsString</strong></td>
<td>Used to specify a method of representation of the BINARY and VARBINARY fields.</td>
</tr>
<tr>
<td><strong>CheckRowVersion</strong></td>
<td>Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CreateConnection</td>
<td>Used to specify if an additional connection to a server should be established to execute an additional query in the TCustomMyDataSet.FetchAll=False mode.</td>
</tr>
<tr>
<td>DefaultValues</td>
<td>Used to fill the DefaultExpression property of TField objects with appropriate value.</td>
</tr>
<tr>
<td>EnableBoolean</td>
<td>Used to specify the method of representation of the TINYINT(1) fields.</td>
</tr>
<tr>
<td>FieldsAsString</td>
<td>Used to store all non-BLOB fields as string (native MySQL format).</td>
</tr>
<tr>
<td>FullRefresh</td>
<td>Used to specify the fields to include in automatically generated SQL statement when calling the TCustomDADataset.RefreshRecord method. Default value is false.</td>
</tr>
<tr>
<td>NullForZeroDate</td>
<td>Used for MySQL server to represent the value for for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).</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 record count for this SELECT so the RecordCount property reflects the actual number of records.</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>Used for TCustomMyDataSet to delete data.</td>
</tr>
</tbody>
</table>
### MySQL Data Access Components

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>quote all field names in autogenerated SQL statements.</strong></td>
<td>Used for dataset to remove a record locally if the RefreshRecord procedure can't find necessary record on the server.</td>
</tr>
<tr>
<td><strong>RemoveOnRefresh</strong></td>
<td>Used for TCustomDADDataSet to set the Required property of TField objects for NOT NULL fields.</td>
</tr>
<tr>
<td><strong>RequiredFields</strong></td>
<td>Used to return the new value of the fields to dataset after insert or update.</td>
</tr>
<tr>
<td><strong>ReturnParams</strong></td>
<td>Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML method.</td>
</tr>
<tr>
<td><strong>SaveBlobsAsHex</strong></td>
<td>Force replace of empty strings with NULL values in data. The default value is False.</td>
</tr>
<tr>
<td><strong>SetEmptyStrToNull</strong></td>
<td>Used to specify whether fields not belonging to the current updating table get read-only attribute.</td>
</tr>
<tr>
<td><strong>SetFieldsReadOnly</strong></td>
<td>Used for TCustomDADDataSet to raise an exception when the number of updated or deleted records does not equal 1.</td>
</tr>
<tr>
<td><strong>TrimFixedChar</strong></td>
<td>Used to specify whether to discard all trailing spaces in string fields of the dataset.</td>
</tr>
<tr>
<td><strong>TrimVarChar</strong></td>
<td>Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.</td>
</tr>
</tbody>
</table>

**See Also**
5.14.1.10.2.1 AutoPrepare Property

Used to execute automatic `TCustomDADataSet.Prepare` on a query execution.

**Class**

`TMyDataSetOptions`

**Syntax**

```plaintext
property AutoPrepare: boolean;
```

**Remarks**

Set the AutoPrepare property to execute automatic `TCustomDADataSet.Prepare` on a query execution. Makes sense for cases when a query will be executed several times, for example, in Master/Detail relationships.

5.14.1.10.2.2 AutoRefresh Property

Used to automatically refresh dataset every `AutoRefreshInterval` seconds.

**Class**

`TMyDataSetOptions`

**Syntax**

```plaintext
property AutoRefresh: boolean default False;
```

**Remarks**

If True, dataset will be automatically refreshed every `AutoRefreshInterval` seconds. If dataset has at least one key field and a `TIMESTAMP` field, the `TCustomMyDataSet.RefreshQuick` method will be executed, otherwise the Refresh method will be executed. This option is only available for Windows.
5.14.1.10.2.3 AutoRefreshInterval Property

Used to define in what time interval in seconds the Refresh or TCustomMyDataSet.RefreshQuick method of a DataSet is called.

Class

TMyDataSetOptions

Syntax

property AutoRefreshInterval: integer default 60;

Remarks

Use the AutoRefreshInterval property to define in what time interval in seconds the Refresh or TCustomMyDataSet.RefreshQuick method of a DataSet is called. This option is only available for Windows.

5.14.1.10.2.4 BinaryAsString Property

Used to specify a method of representation of the BINARY and VARBINARY fields.

Class

TMyDataSetOptions

Syntax

property BinaryAsString: boolean default True;

Remarks

Use the BinaryAsString property to specify a method of representation of the BINARY and VARBINARY fields. If set to True, these fields will be represented as string fields; otherwise, as TBytesField and TVarBytesField correspondingly. If the binary fields should not be processed as strings then set this property to False. The default value is True.
5.14.1.10.2.5 CheckRowVersion Property

Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.

Class
TMyDataSetOptions

Syntax

```
property CheckRowVersion: boolean default False;
```

Remarks
Use the CheckRowVersion property to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data. If CheckRowVersion is True and DataSet has timestamp field when only this field is added into WHERE clause of generated SQL statement. If CheckRowVersion is True, but there is no TIMESTAMP field, then all nonblob fields will be added to WHERE clause. The default value is False.

5.14.1.10.2.6 CreateConnection Property

Used to specify if an additional connection to a server should be established to execute an additional query in the TCustomMyDataSet.FetchAll=False mode.

Class
TMyDataSetOptions

Syntax

```
property CreateConnection: boolean default True;
```

Remarks
Use the CreateConnection property to specify if an additional connection to a server should be established to execute an additional query in the TCustomMyDataSet.FetchAll=False mode.
mode. If a Dataset is opened in `TCustomMyDataSet.FetchAll=False`, the current connection is blocked until all records have been fetched. If this option is set to True, an additional connection is created to fetch data to avoid blocking of the current connection.

5.14.1.10.2.7 DefaultValues Property

Used to fill the DefaultExpression property of TField objects with appropriate value.

Class

`TMyDataSetOptions`

Syntax

```
property DefaultValues: boolean;
```

Remarks

If True, TCustomMyDataSet fills the DefaultExpression property of TField objects with appropriate value.

5.14.1.10.2.8 EnableBoolean Property

Used to specify the method of representation of the TINYINT(1) fields.

Class

`TMyDataSetOptions`

Syntax

```
property EnableBoolean: boolean default True;
```

Remarks

Use the EnableBoolean property to specify the method of representation of the TINYINT(1) fields. If set to True, these fields will be represented as TBooleanField; otherwise, as TSmallintField. The default value is True.
5.14.10.2.9 FieldsAsString Property

Used to store all non-BLOB fields as string (native MySQL format).

Class

TMyDataSetOptions

Syntax

property FieldsAsString: boolean default False;

Remarks

All non-BLOB fields are stored as string (native MySQL format). The default value is False.

5.14.10.2.10 FullRefresh Property

Used to specify the fields to include in automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. Default value is false.

Class

TMyDataSetOptions

Syntax

property FullRefresh: boolean;

Remarks

Use the FullRefresh property to specify what fields to include in automatically generated SQL statement when calling the TCustomDADataSet.RefreshRecord method. If the FullRefresh property is True, all the fields from query are included into a SQL statement to refresh single record. If FullRefresh is False, only fields from TMyQuery.UpdatingTable are included. The default value is False.
5.14.1.10.2.11 NullForZeroDate Property

Used for MySQL server to represent the value for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).

Class

TMyDataSetOptions

Syntax

```
property NullForZeroDate: boolean default True;
```

Remarks

For datetime fields with invalid values, for example '2002-12-32', MySQL returns on fetch the '0000-00-00' value. According to the NullForZeroDate option this value will be represented as Null or '0001-01-01' ('0100-01-01' for CLR). The default value is True.

© 1997-2021
Devart. All Rights Reserved.

5.14.1.10.2.12 NumberRange Property

Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Class

TMyDataSetOptions

Syntax

```
property NumberRange: boolean;
```

Remarks

Use the NumberRange property to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values. The default value is False.

© 1997-2021
Devart. All Rights Reserved.
5.14.1.10.2.13 QueryRecCount Property

Used for TCustomDADataSet to perform additional query to get record count for this SELECT so the RecordCount property reflects the actual number of records.

Class

TMyDataSetOptions

Syntax

property QueryRecCount: boolean;

Remarks

If True, and the TCustomMyDataSet.FetchAll property is False, TCustomDADataSet performs additional query to get record count for this SELECT so the RecordCount property reflects the actual number of records. Does not have any effect if the TCustomMyDataSet.FetchAll property is True.

5.14.1.10.2.14 QuoteNames Property

Used for TCustomMyDataSet to quote all field names in autogenerated SQL statements.

Class

TMyDataSetOptions

Syntax

property QuoteNames: boolean;

Remarks

If True, TCustomMyDataSet quotes all field names in autogenerated SQL statements such as update SQL.
5.14.1.10.2.15 RemoveOnRefresh Property

Used for dataset to remove a record locally if the RefreshRecord procedure can't find necessary record on the server.

Class

TMyDataSetOptions

Syntax

property RemoveOnRefresh: boolean;

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 its key value.

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.

5.14.1.10.2.16 RequiredFields Property

Used for TCustomDADataSet to set the Required property of TField objects for NOT NULL fields.

Class

TMyDataSetOptions

Syntax

property RequiredFields: boolean default False;

Remarks

If True, TCustomDADataSet sets the Required property of TField objects for NOT NULL fields. It is useful when table has a trigger that updates NOT NULL fields. The default value is False.
5.14.1.10.2.17 ReturnParams Property

Used to return the new value of the fields to dataset after insert or update.

Class

TMyDataSetOptions

Syntax

```plaintext
property ReturnParams: boolean;
```

Remarks

Use the ReturnParams property to return the new value of the fields to dataset after insert or update. Actual value of field after insert or update may be different from the value stored in local memory if the table has a trigger. When ReturnParams is True, OUT parameters of SQLInsert and SQLUpdate statements is assigned to corresponding fields. The default value is False.

5.14.1.10.2.18 SaveBlobsAsHex Property

Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML method.

Class

TMyDataSetOptions

Syntax

```plaintext
property SaveBlobsAsHex: boolean default True;
```

Remarks

Use the SaveBlobsAsHex property to save BLOB data in hexadecimal format when calling the TMyQuery.SaveToXML method. If the SaveBlobsAsHex property is set to False, the value of the BLOB field is saved as a string. The default value is True.
5.14.1.10.2.19 SetEmptyStrToNull Property

Force replace of empty strings with NULL values in data. The default value is False.

Class

TMyDataSetOptions

Syntax

```
property SetEmptyStrToNull: boolean;
```

5.14.1.10.2.20 SetFieldsReadOnly Property

Used to specify whether fields not belonging to the current updating table get read-only attribute.

Class

TMyDataSetOptions

Syntax

```
property SetFieldsReadOnly: boolean default False;
```

Remarks

Use the SetFieldsReadOnly property to specify whether fields not belonging to the current updating table get read-only attribute. The default value is False.

5.14.1.10.2.21 StrictUpdate Property

Used for TCustomDADataset to raise an exception when the number of updated or deleted records does not equal 1.

Class
**TMyDataSetOptions**

**Syntax**

```pascal
property StrictUpdate: boolean;
```

**Remarks**

TCustomDADataset raises an exception when the number of updated or deleted records does not equal 1. Setting this option also causes an exception if the RefreshRecord procedure returns more than one record. The exception does not occur when you use a non-SQL block. The default value is True.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.10.2.22 TrimFixedChar Property

Used to specify whether to discard all trailing spaces in string fields of the dataset.

**Class**

*TMyDataSetOptions*

**Syntax**

```pascal
property TrimFixedChar: boolean;
```

**Remarks**

Use the TrimFixedChar property to specify whether to discard all trailing spaces in string fields of the dataset. The default value is True.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.10.2.23 TrimVarChar Property

Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.

**Class**

*TMyDataSetOptions*
Syntax

```pascal
property TrimVarChar: boolean;
```

Remarks

Use the TrimVarChar property to specify whether to discard all trailing spaces in the variable-length string fields of a dataset. The default value is False.

5.14.1.11 TMyDataSource Class

TMyDataSource provides an interface between a MyDAC dataset components and data-aware controls on a form.

For a list of all members of this type, see TMyDataSource members.

Unit

`MyAccess`

Syntax

```pascal
TMyDataSource = class(TCRDataSource);
```

Remarks

TMyDataSource provides an interface between a MyDAC dataset components and data-aware controls on a form.

TMyDataSource 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`
- `TMyDataSource`
5.14.1.11 Members

**TMyDataSource** class overview.

© 1997-2021 Devart. All Rights Reserved.

5.14.1.12 TMyEncryptor Class

The class that performs encrypting and decrypting of data.

For a list of all members of this type, see **TMyEncryptor** members.

Unit

**MyAccess**

Syntax

```pascal
TMyEncryptor = class(TCREncryptor);
```

Inheritance Hierarchy

TCREncryptor

  TMyEncryptor

© 1997-2021 Devart. All Rights Reserved.

5.14.1.12.1 Members

**TMyEncryptor** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataHeader</strong> (inherited from <strong>TCREncryptor</strong>)</td>
<td>Specifies whether the additional information is stored with the encrypted data.</td>
</tr>
<tr>
<td><strong>EncryptionAlgorithm</strong> (inherited from <strong>TCREncryptor</strong>)</td>
<td>Specifies the algorithm of data encryption.</td>
</tr>
<tr>
<td><strong>HashAlgorithm</strong> (inherited from <strong>TCREncryptor</strong>)</td>
<td>Specifies the algorithm of generating hash data.</td>
</tr>
<tr>
<td><strong>InvalidHashAction</strong> (inherited from <strong>TCREncryptor</strong>)</td>
<td>Specifies the action to</td>
</tr>
</tbody>
</table>
### MySQL Data Access Components

**Password** (inherited from **TCREncryptor**)  
Used to set a password that is used to generate a key for encryption.

---

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>SetKey</em> (inherited from <strong>TCREncryptor</strong>)</td>
<td>Sets a key, using which data is encrypted.</td>
</tr>
</tbody>
</table>

---

### 5.14.1.13 TMyMetaData Class

A component for obtaining metainformation about database objects from the server.

For a list of all members of this type, see **TMyMetaData** members.

**Unit**  
**MyAccess**

**Syntax**

```delphi
TMyMetaData = class(TDAMetaData);
```

**Remarks**

The TMyMetaData component is used to obtain metainformation from the server about objects in the database, such as tables, table columns, stored procedures, etc. TMyMetaData publishes properties of **TDAMetaData**.

**Inheritance Hierarchy**

- **TMemDataSet**  
- **TDAMetaData**  
- **TMyMetaData**

**See Also**

- **TCustomDADataset.Debug**
5.14.1.13.1 Members

**TMyMetaData** class overview.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>Connection</td>
<td>(inherited from <strong>TDAMetaData</strong>) Used to specify a connection object to use to connect to a data store.</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>KeyExclusive</td>
<td>(inherited from <strong>TMemDataSet</strong>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>LocalConstraints</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>LocalUpdate</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MetaDataKind</td>
<td>(inherited from <strong>TDAMetaData</strong>) Used to specify which kind of metainformation to show.</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>Restrictions</td>
<td>(inherited from <strong>TDAMetaData</strong>) Used to provide one or more conditions restricting the list of objects to be described.</td>
</tr>
<tr>
<td>UpdateRecordTypes</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to indicate the update</td>
</tr>
</tbody>
</table>
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><strong>GetRestrictions</strong> (inherited from <strong>TDAMetaData</strong>)</td>
<td>Used to find out which restrictions are applicable to a certain MetaDataKind.</td>
</tr>
<tr>
<td><strong>Locate</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Overloaded. Searches a dataset for a specific record.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</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. and positions the cursor on it.</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>
<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>
</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 <strong>TMemDataSet</strong>)</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
<tr>
<td>OnUpdateRecord (inherited from <strong>TMemDataSet</strong>)</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

5.14.1.14 **TMyQuery 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 **TMyQuery** members.

**Unit**

**MyAccess**

**Syntax**

```
TMyQuery = class(TCustomMyDataSet);
```

**Remarks**

TMyQuery is a direct descendant of the **TCustomMyDataSet** component. It publishes most of its inherited properties and events so that they can be manipulated at design-time.

Use TMyQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. TMyQuery 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, TMyQuery will retrieve primary keys for UpdatingTable from metadata. TMyQuery 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
  - TCustomMyDataSet
  - TMyQuery

See Also
- Updating Data with MyDAC Dataset Components
- Master/Detail Relationships
- TMyStoredProc
- TMyTable

© 1997-2021
Devart. All Rights Reserved.

5.14.1.14.1 Members

**TMyQuery** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSQL</td>
<td>(inherited from TCustomDADataSet) Used to return SQL text without any changes</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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>CommandTimeout</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Used to specify the amount of time to attempt execution of a command.</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>TCustomMyDataSet</strong>) Used to specify a connection object that will be used to connect to a data store.</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>Encryption</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Used to specify encryption options in a dataset.</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.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>IndexFieldNames</code></td>
<td>(inherited from <code>TMemDataSet</code>) Used to get or set the list of fields on which the recordset is sorted.</td>
</tr>
<tr>
<td><code>InsertId</code></td>
<td>(inherited from <code>TCustomMyDataSet</code>) Returns the ID generated for an AUTO_INCREMENT column by the previous query.</td>
</tr>
<tr>
<td><code>IsQuery</code></td>
<td>(inherited from <code>TCustomDADataSet</code>) Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td><code>KeyExclusive</code></td>
<td>(inherited from <code>TMemDataSet</code>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><code>KeyFields</code></td>
<td>(inherited from <code>TCustomDADataSet</code>) Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</td>
</tr>
<tr>
<td><code>LocalConstraints</code></td>
<td>(inherited from <code>TMemDataSet</code>) 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></td>
<td>(inherited from <code>TMemDataSet</code>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><code>LockMode</code></td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
</tr>
<tr>
<td><code>MacroCount</code></td>
<td>(inherited from <code>TCustomDADataSet</code>) Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td><code>Macros</code></td>
<td>(inherited from <code>TCustomDADataSet</code>) Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td><code>MasterFields</code></td>
<td>(inherited from <code>TCustomDADataSet</code>) 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>
</tbody>
</table>

AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
<table>
<thead>
<tr>
<th>Property</th>
<th>Inherited From</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MasterSource</td>
<td>TCustomDADataSet</td>
<td>Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td>TCustomMyDataSet</td>
<td>Specifies the behaviour of TCustomMyDataSet object.</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>TCustomDADataSet</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>TCustomDADataSet</td>
<td>Used to indicate how many parameters are there in the Params property.</td>
</tr>
<tr>
<td>Params</td>
<td>TCustomDADataSet</td>
<td>Used to view and set parameter names, values, and data types dynamically.</td>
</tr>
<tr>
<td>Prepared</td>
<td>TMemDataSet</td>
<td>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td>Ranged</td>
<td>TMemDataSet</td>
<td>Indicates whether a range is applied to a dataset.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>TCustomDADataSet</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td>RefreshOptions</td>
<td>TCustomDADataSet</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td>RowsAffected</td>
<td>TCustomDADataSet</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SmartFetch</td>
<td>TCustomMyDataSet</td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</td>
</tr>
<tr>
<td>SQL</td>
<td>TCustomDADataSet</td>
<td>Used to provide a SQL statement that a query component executes when its Open component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete</td>
<td>TCustomDADataSet</td>
<td>Used to specify a SQL statement that will be used when applying a deletion to a record.</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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>UpdateObject (inherited from TCustomMyDataSet)</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
<td></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>
<td></td>
</tr>
<tr>
<td>UpdatesPending (inherited from TMemDataSet)</td>
<td>Used to check the status of the cached updates buffer.</td>
<td></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 records.</td>
<td></td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddWhere</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<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>BreakExec</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Breaks execution of the SQL statement on the server.</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>CreateBlobStream</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) 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>DeferredPost</strong></td>
<td>(inherited from <strong>TMemDataset</strong>) Makes permanent changes to the database server.</td>
</tr>
<tr>
<td><strong>DeleteWhere</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Removes WHERE clause from the SQL property and assigns the BaseSQL property.</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>Execute</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Overloaded. Executes a SQL statement on the server.</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>TCustomDADataSet</td>
</tr>
<tr>
<td>GetBlob</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>GetDataType</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetFieldEnum</td>
<td>TCustomMyDataSet</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>Method Name</td>
<td>Inherited From</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>TCustomDADataSet</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>TCustomMyDataSet</td>
</tr>
<tr>
<td>LockTable</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>MacroByName</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>OpenNext</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>ParamByName</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Prepare</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>RefreshQuick</td>
<td>TCustomMyDataSet</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>RevertRecord (inherited from TMemDataSet)</td>
<td>Unapplied.</td>
</tr>
<tr>
<td>SaveSQL (inherited from TCustomDADataset)</td>
<td>Saves the SQL property value to BaseSQL.</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>SetOrderBy (inherited from TCustomDADataset)</td>
<td>Builds an ORDER BY clause of a SELECT statement.</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>SQLSaved (inherited from TCustomDADataset)</td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td>UnLock (inherited from TCustomDADataset)</td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td>UnLockTable (inherited from TCustomMyDataset)</td>
<td>Releases a table locked by the TCustomMyDataset.LockTable method.</td>
</tr>
<tr>
<td>UnPrepare (inherited from TMemDataSet)</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult (inherited from TMemDataSet)</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.

### 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 insert, delete, update, lock, and refresh operations.</td>
</tr>
<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>

© 1997-2021 Devart. All Rights Reserved.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CachedUpdates</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>CommandTimeout</strong></td>
<td>(inherited from <code>TCustomMyDataSet</code>) Used to specify the amount of time to attempt execution of a command.</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>TCustomMyDataSet</code>) Used to specify a connection object that will be used to connect to a data store.</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 MasterFields 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>Encryption</strong></td>
<td>(inherited from <code>TCustomMyDataSet</code>) Used to specify encryption options in a dataset.</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 expanded macros.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</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>InsertId</td>
<td>(inherited from TCustomMyDataSet) Returns the ID generated for an AUTO_INCREMENT column by the previous query.</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>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>(inherited from TCustomDADataSet) Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>(inherited from TCustomDADataSet) Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</td>
<td>(inherited from TCustomDADataSet) 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>(inherited from TCustomDADataSet) Used to specify the data source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options</td>
<td>(inherited from TCustomMyDataSet) Specifies the behaviour of TCustomMyDataSet object.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</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>TCustomDADataset</strong>) 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>(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 editing record is refreshed.</td>
</tr>
<tr>
<td><strong>RowsAffected</strong></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><strong>SmartFetch</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</td>
</tr>
<tr>
<td><strong>SQL</strong></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><strong>SQLDelete</strong></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><strong>SQLInsert</strong></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><strong>SQLLock</strong></td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify a SQL statement that will be used</td>
</tr>
</tbody>
</table>
### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 updates.</td>
</tr>
</tbody>
</table>
data-modification queries as a result of user incentive to insert, update or delete records.

See Also
- TMyQuery Class
- TMyQuery Class Members

Class
TMyQuery

Syntax

```property``
``FetchAll: boolean;``

Remarks

Defines whether to request all records of the query from database server when the dataset is being opened.

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.

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.
5.14.1.14.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

TMyQuery

Syntax

```pascal
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

- TMyStoredProc.LockMode
- TMyTable.LockMode

5.14.1.14.2.3 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

TMyQuery

Syntax

```pascal
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 TCustomMyDataSet.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. All fields from other than target table have their ReadOnly properties set to True (if TCustomMyDataSet.Options).

Example

The first example specifies a query in which only one value 'Dept' for the UpdatingTable property is allowed.

The second example shows a query in which allowed values for UpdatingTable are 'Dept' and 'Emp'.

By default the updating table will be the first used table, so 'DEPT' and all fields of DEPT will be editable. If however UpdatingTable is set to be 'EMP' all fields of EMP will be editable.

Example 1.

```
SELECT * FROM Dept
```

Example 2.

```
SELECT * FROM Dept, Emp
WHERE Dept.DeptNo = Emp.DeptNo
```

5.14.1.15 TMyStoredProc Class

A component for accessing and executing stored procedures and functions.

For a list of all members of this type, see TMyStoredProc members.

Unit

MyAccess

Syntax

```
TMyStoredProc = class(TCustomMyStoredProc);
```

Remarks
Use TMyStoredProc 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 return parameters in the Params property.

**Inheritance Hierarchy**

- `TMemDataSet`
  - `TCustomDADataSet`
    - `TCustomMyDataSet`
      - `TCustomMyStoredProc`
        - `TMyStoredProc`

**See Also**
- `TMyQuery`
- `TMyCommand`
- `Updating Data with MyDAC Dataset Components`

© 1997-2021 Devart. All Rights Reserved.

5.14.1.15.1 Members

**TMyStoredProc** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BaseSQL</code> (inherited from <code>TCustomDADataSet</code>)</td>
<td>Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.</td>
</tr>
<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>CommandTimeout</code> (inherited from <code>TCustomMyDataSet</code>)</td>
<td>Used to specify the amount of time to attempt execution of a command.</td>
</tr>
<tr>
<td><code>Conditions</code> (inherited from <code>TCustomDADataSet</code>)</td>
<td>Used to add WHERE conditions to a query.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>DetailFields</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>Disconnected</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>Encryption</strong></td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td><strong>FetchAll</strong></td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td><strong>FetchRows</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>FilterSQL</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>FinalSQL</strong></td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>TMemDataSet</td>
</tr>
<tr>
<td><strong>InsertId</strong></td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>IsQuery</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Used to check whether SQL statement returns rows.</td>
</tr>
<tr>
<td><strong>KeyExclusive</strong> (inherited from <strong>TMemDataSet</strong>)</td>
<td>Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>KeyFields</strong> (inherited from <strong>TCustomDADataSet</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>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 implicit update of rows on database server.</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>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> (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Specifies the behaviour of TCustomMyDataSet 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>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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>TCustomDADataSet</strong>)</td>
<td>Used to view and set parameter names, values, and data types dynamically.</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>SmartFetch</strong> (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</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 the record count when opening a dataset.</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</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</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 data to a BLOB field, specified by the Field parameter.</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</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------</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>TCustomDADataSet</td>
</tr>
<tr>
<td></td>
<td></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></td>
<td></td>
</tr>
<tr>
<td>GetFieldEnum</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>GetFieldObject</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td></td>
<td></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></td>
<td></td>
</tr>
<tr>
<td>GetOrderBy</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>GotoCurrent</td>
<td>TCustomDADataSet</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>TCustomMyDataSet</td>
</tr>
<tr>
<td>LockTable</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>MacroByName</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>OpenNext</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>ParamByName</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Prepare</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>PrepareSQL</td>
<td>TCustomMyStoredProc</td>
</tr>
<tr>
<td>RefreshQuick</td>
<td>TCustomMyDataSet</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>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>UnLockTable</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>TMemDataSet</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><strong>AfterExecute</strong></td>
<td>(inherited from <a href="#">TCustomDADataSet</a>) Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td><strong>AfterFetch</strong></td>
<td>(inherited from <a href="#">TCustomDADataSet</a>) Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td><strong>AfterUpdateExecute</strong></td>
<td>(inherited from <a href="#">TCustomDADataSet</a>) Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td><strong>BeforeFetch</strong></td>
<td>(inherited from <a href="#">TCustomDADataSet</a>) Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td><strong>BeforeUpdateExecute</strong></td>
<td>(inherited from <a href="#">TCustomDADataSet</a>) Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td><strong>OnUpdateError</strong></td>
<td>(inherited from <a href="#">TMemDataSet</a>) 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 <a href="#">TMemDataSet</a>) Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.14.1.15.2 Properties

Properties of the **TMyStoredProc** class.

For a complete list of the **TMyStoredProc** class members, see the [TMyStoredProc Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

<p>| <strong>BaseSQL</strong> (inherited from <strong>TCustomDADataset</strong>) | Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL. |
| <strong>CachedUpdates</strong> (inherited from <strong>TMemDataSet</strong>) | Used to enable or disable the use of cached updates for a dataset. |
| <strong>CommandTimeout</strong> (inherited from <strong>TCustomMyDataSet</strong>) | Used to specify the amount of time to attempt execution of a command. |
| <strong>Conditions</strong> (inherited from <strong>TCustomDADataset</strong>) | Used to add WHERE conditions to a query. |
| <strong>Connection</strong> (inherited from <strong>TCustomMyDataSet</strong>) | Used to specify a connection object that will be used to connect to a data store. |
| <strong>DataTypeMap</strong> (inherited from <strong>TCustomDADataset</strong>) | Used to set data type mapping rules. |
| <strong>Debug</strong> (inherited from <strong>TCustomDADataset</strong>) | Used to display the statement that is being executed and the values and types of its parameters. |
| <strong>DetailFields</strong> (inherited from <strong>TCustomDADataset</strong>) | Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship. |
| <strong>Disconnected</strong> (inherited from <strong>TCustomDADataset</strong>) | Used to keep dataset opened after connection is closed. |
| <strong>Encryption</strong> (inherited from <strong>TCustomMyDataSet</strong>) | Used to specify encryption options in a dataset. |
| <strong>FetchAll</strong> (inherited from <strong>TCustomMyDataSet</strong>) | Defines whether to request all records of the query from database server when the dataset is being opened. |
| <strong>FetchRows</strong> (inherited from <strong>TCustomDADataset</strong>) | Used to define the number of rows to be transferred across the network at the same time. |
| <strong>FilterSQL</strong> (inherited from <strong>TCustomDADataset</strong>) | Used to change the WHERE clause of SELECT statement and reopen a query. |
| <strong>FinalSQL</strong> (inherited from <code>TCustomDADataSet</code>) | Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros. |
| <strong>IndexFieldNames</strong> (inherited from <code>TMemDataSet</code>) | Used to get or set the list of fields on which the recordset is sorted. |
| <strong>InsertId</strong> (inherited from <code>TCustomMyDataSet</code>) | Returns the ID generated for an AUTO_INCREMENT column by the previous query. |
| <strong>IsQuery</strong> (inherited from <code>TCustomDADataSet</code>) | Used to check whether SQL statement returns rows. |
| <strong>KeyExclusive</strong> (inherited from <code>TMemDataSet</code>) | Specifies the upper and lower boundaries for a range. |
| <strong>KeyFields</strong> (inherited from <code>TCustomDADataSet</code>) | Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database. |
| <strong>LocalConstraints</strong> (inherited from <code>TMemDataSet</code>) | Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet. |
| <strong>LocalUpdate</strong> (inherited from <code>TMemDataSet</code>) | Used to prevent implicit update of rows on database server. |
| <strong>MacroCount</strong> (inherited from <code>TCustomDADataSet</code>) | Used to get the number of macros associated with the Macros property. |
| <strong>Macros</strong> (inherited from <code>TCustomDADataSet</code>) | Makes it possible to change SQL queries easily. |
| <strong>MasterFields</strong> (inherited from <code>TCustomDADataSet</code>) | 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>MasterSource</strong> (inherited from <code>TCustomDADataSet</code>) | Used to specify the data source component which |</p>
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options</strong></td>
<td>Specifies the behaviour of TCustomMyDataSet 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>Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></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>SmartFetch</strong></td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</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</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQLLock (inherited from TCustomDADataSet)</td>
<td>when applying an insertion to a dataset.</td>
</tr>
<tr>
<td>SQLRecCount (inherited from TCustomDADataSet)</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRefresh (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>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 (inherited from TCustomMyStoredProc)</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 TCustomMyDataSet)</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the 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>

**Published**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LockMode</td>
<td>Used to specify what kind of lock will be performed when editing a record.</td>
</tr>
</tbody>
</table>
UpdatingTable

Specifies 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.

See Also

- TMyStoredProc Class
- TMyStoredProc Class Members

5.14.1.15.2.1 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

TMyStoredProc

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

- TMyQuery.LockMode
- TMyTable.LockMode
5.14.1.15.2.2 UpdatingTable Property

Specifies 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

TMyStoredProc

Syntax

```
property UpdatingTable: string;
```

Remarks

the UpdatingTable property is used on Insert, Update, Delete or RefreshRecord (see also TCustomMyDataSet.Options) if appropriate SQL (SQLInsert, SQLUpdate or SQLDelete) is not provided.

If UpdatingTable is not set then the first table used in query is assumed to be the target.

If the query is addressed to the View then entire View is taken as a target for subsequent modifications.

All fields from other than target table have their ReadOnly properties set to True (if TCustomMyDataSet.Options is True).

© 1997-2021
Devart. All Rights Reserved.

5.14.1.16 TMyTable 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 TMyTable members.

Unit

MyAccess

Syntax

```
TMyTable = class(TCustomMyTable);
```

Remarks

The TMyTable component allows retrieving and updating data in a single table without writing
SQL statements. Use TMyTable to access data in a table. Use the TableName property to specify table name. TMyTable 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
    - TCustomMyDataSet
    - TCustomMyTable
  - TMyTable

See Also
- [Updating Data with MyDAC Dataset Components](#)
- [Master/Detail Relationships](#)
- [TCustomMyDataSet](#)
- [TMyQuery](#)
- [TCustomMyTable](#)

© 1997-2021 Devart. All Rights Reserved.

---

**TMyTable** 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 (inherited from TCustomMyDataSet)</td>
<td>Used to specify the amount of time to attempt execution of a command.</td>
</tr>
<tr>
<td>Conditions (inherited from TCustomDADataset)</td>
<td>Used to add WHERE</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Used to specify a connection object that will be used to connect to a data store.</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>Encryption</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Used to specify encryption options in a dataset.</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>IndexDefs</strong></td>
<td>(inherited from <strong>TCustomMyTable</strong>) Contains information about the indexes for a table.</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>InsertId</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Returns the ID generated for</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsQuery</td>
<td>Checks whether SQL statement returns rows.</td>
</tr>
<tr>
<td>KeyExclusive</td>
<td>Specifies upper and lower boundaries for a range.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>Builds SQL statements for SQLDelete, SQLInsert, and SQLUpdate properties.</td>
</tr>
<tr>
<td>Limit</td>
<td>Sets number of rows retrieved from the query.</td>
</tr>
<tr>
<td>LocalConstraints</td>
<td>Avoids setting the Required property of a TField component for NOT NULL fields.</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>Prevents implicit update of rows on database server.</td>
</tr>
<tr>
<td>LockMode</td>
<td>Specifies what kind of lock will be performed editing a record.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>Gets 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>Specifies names of fields used as foreign keys for dataset.</td>
</tr>
<tr>
<td>MasterSource</td>
<td>Specifies data source component which binds current dataset to master.</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>Specifies the behaviour of the TMyTable 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>Used to view and set parameter names, values, and data types dynamically.</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, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SmartFetch</td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</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</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLInsert (inherited from <code>TCustomDADataset</code>)</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 <code>TCustomDADataset</code>)</td>
<td>Used to specify a SQL statement that will be used to perform a record lock.</td>
</tr>
<tr>
<td>SQLRecCount (inherited from <code>TCustomDADataset</code>)</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 <code>TCustomDADataset</code>)</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>SQLUpdate (inherited from <code>TCustomDADataset</code>)</td>
<td>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 (inherited from <code>TCustomDADataset</code>)</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>UpdateObject (inherited from <code>TCustomMyDataSet</code>)</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
</tr>
<tr>
<td>UpdateRecordTypes (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>UpdatesPending (inherited from <code>TMemDataSet</code>)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td><strong>AddWhere</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>ApplyRange</strong></td>
<td><code>TMemDataSet</code></td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong></td>
<td><code>TMemDataSet</code></td>
</tr>
<tr>
<td><strong>BreakExec</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>CancelRange</strong></td>
<td><code>TMemDataSet</code></td>
</tr>
<tr>
<td><strong>CancelUpdates</strong></td>
<td><code>TMemDataSet</code></td>
</tr>
<tr>
<td><strong>CommitUpdates</strong></td>
<td><code>TMemDataSet</code></td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>DeferredPost</strong></td>
<td><code>TMemDataSet</code></td>
</tr>
<tr>
<td><strong>DeleteWhere</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
<tr>
<td><strong>EditRangeEnd</strong></td>
<td><code>TMemDataSet</code></td>
</tr>
<tr>
<td><strong>EditRangeStart</strong></td>
<td><code>TMemDataSet</code></td>
</tr>
<tr>
<td><strong>EmptyTable</strong></td>
<td><code>TCustomMyTable</code></td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td><code>TCustomDADataset</code></td>
</tr>
</tbody>
</table>
| **Executing**   | `TCustomDADataset`      | Indicates whether SQL
<table>
<thead>
<tr>
<th>Function</th>
<th>Inherited From</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetched</td>
<td><code>TCustomDADataSet</code></td>
<td>Used to find out whether <code>TCustomDADataSet</code> has fetched all rows.</td>
</tr>
<tr>
<td>Fetching</td>
<td><code>TCustomDADataSet</code></td>
<td>Used to learn whether <code>TCustomDADataSet</code> is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll</td>
<td><code>TCustomDADataSet</code></td>
<td>Used to learn whether <code>TCustomDADataSet</code> is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey</td>
<td><code>TCustomDADataSet</code></td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro</td>
<td><code>TCustomDADataSet</code></td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest</td>
<td><code>TCustomDADataSet</code></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><code>TCustomDADataSet</code></td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>GetBlob</td>
<td><code>TMemDataSet</code></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><code>TCustomDADataSet</code></td>
<td>Returns internal field types defined in the MemData and accompanying modules.</td>
</tr>
<tr>
<td>GetFieldEnum</td>
<td><code>TCustomMyDataSet</code></td>
<td>Retrieve the list of acceptable values for a specified field given by the FieldName parameter.</td>
</tr>
<tr>
<td>GetFieldObject</td>
<td><code>TCustomDADataSet</code></td>
<td>Returns a multireference shared object from field.</td>
</tr>
<tr>
<td>GetFieldPrecision</td>
<td><code>TCustomDADataSet</code></td>
<td>Retrieves the precision of a number field.</td>
</tr>
<tr>
<td>GetFieldScale</td>
<td><code>TCustomDADataSet</code></td>
<td>Retrieves the scale of a number field.</td>
</tr>
<tr>
<td>GetKeyFieldNames</td>
<td></td>
<td>Provides a list of available</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>TCustomDADataSet</strong></td>
<td>key field names.</td>
<td></td>
</tr>
<tr>
<td>GetOrderBy (inherited from TCustomDADataSet)</td>
<td>Retrieves an ORDER BY clause from a SQL statement.</td>
<td></td>
</tr>
<tr>
<td>GotoCurrent (inherited from TCustomDADataSet)</td>
<td>Sets the current record in this dataset similar to the current record in another dataset.</td>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>Lock (inherited from TCustomMyDataSet)</td>
<td>Overloaded. Locks the current record for the current connection.</td>
<td></td>
</tr>
<tr>
<td>LockTable (inherited from TCustomMyDataSet)</td>
<td>Locks table for the current connection.</td>
<td></td>
</tr>
<tr>
<td>MacroByName (inherited from TCustomDADataSet)</td>
<td>Finds a macro with the specified name.</td>
<td></td>
</tr>
<tr>
<td>OpenNext (inherited from TCustomMyDataSet)</td>
<td>Opens next rowset in the statement.</td>
<td></td>
</tr>
<tr>
<td>ParamByName (inherited from TCustomDADataSet)</td>
<td>Sets or uses parameter information for a specific parameter based on its name.</td>
<td></td>
</tr>
<tr>
<td>Prepare (inherited from TCustomDADataSet)</td>
<td>Allocates, opens, and parses cursor for a query.</td>
<td></td>
</tr>
<tr>
<td>RefreshQuick (inherited from TCustomMyDataSet)</td>
<td>Retrieves changes posted to the server by another clients on the client side quickly.</td>
<td></td>
</tr>
<tr>
<td>RefreshRecord (inherited from TCustomDADataSet)</td>
<td>Actualizes field values for the current record.</td>
<td></td>
</tr>
<tr>
<td>RestoreSQL (inherited from TCustomDADataSet)</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</td>
<td></td>
</tr>
<tr>
<td>RestoreUpdates (inherited from TMemDataSet)</td>
<td>Marks all records in the cache of updates as unapplied.</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Inherited From</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</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>
<tr>
<td>SQLSaved</td>
<td>TCustomDADataset</td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td>UnLock</td>
<td>TCustomDADataset</td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td>UnLockTable</td>
<td>TCustomMyDataSet</td>
<td>Releases a table locked by the TCustomMyDataSet.LockTable method.</td>
</tr>
<tr>
<td>UnPrepare</td>
<td>TMemDataSet</td>
<td>Frees the resources allocated for a previously prepared query on the server and client sides.</td>
</tr>
<tr>
<td>UpdateResult</td>
<td>TMemDataSet</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.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</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 and refresh operations.</td>
</tr>
<tr>
<td><strong>BeforeFetch</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td><strong>BeforeUpdateExecute</strong> (inherited from <strong>TCustomDADataSet</strong>)</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<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>

5.14.1.16.2 Properties

Properties of the **TMyTable** class.

For a complete list of the **TMyTable** class members, see the **TMyTable 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</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</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>CommandTimeout</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Used to specify the amount of time to attempt execution of a command.</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>TCustomMyDataSet</strong>) Used to specify a connection object that will be used to connect to a data store.</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>Encryption</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Used to specify encryption options in a dataset.</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>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IndexDefs</td>
<td>(inherited from <strong>TCustomMyTable</strong>) Contains information about the indexes for a table.</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>InsertId</td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Returns the ID generated for an AUTO_INCREMENT column by the previous query.</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>Limit</td>
<td>(inherited from <strong>TCustomMyTable</strong>) Used to set the number of rows retrieved from the query.</td>
</tr>
<tr>
<td>LocalConstraints</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>LocalUpdate</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td>MacroCount</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to get the number of macros associated with the Macros property.</td>
</tr>
<tr>
<td>Macros</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Makes it possible to change SQL queries easily.</td>
</tr>
<tr>
<td>MasterFields</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>MasterSource</td>
<td>(inherited from <strong>TCustomDADataset</strong>) Used to specify the data.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Offset (inherited from TCustomMyTable)</td>
<td>Source component which binds current dataset to the master one.</td>
</tr>
<tr>
<td>Options (inherited from TCustomMyTable)</td>
<td>Used to allow retrieving data from the server starting from the specified row.</td>
</tr>
<tr>
<td>ParamCheck (inherited from TCustomDADataset)</td>
<td>Specifies the behaviour of the TMyTable object.</td>
</tr>
<tr>
<td>ParamCount (inherited from TCustomDADataset)</td>
<td>Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.</td>
</tr>
<tr>
<td>Params (inherited from TCustomDADataset)</td>
<td>Used to indicate how many parameters are there in the Params property.</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>ReadOnly (inherited from TCustomDADataset)</td>
<td>Used to prevent users from updating, inserting, or deleting data in the dataset.</td>
</tr>
<tr>
<td>RefreshOptions (inherited from TCustomDADataset)</td>
<td>Used to indicate when the editing record is refreshed.</td>
</tr>
<tr>
<td>RowsAffected (inherited from TCustomDADataset)</td>
<td>Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.</td>
</tr>
<tr>
<td>SmartFetch (inherited from TCustomMyDataSet)</td>
<td>The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.</td>
</tr>
<tr>
<td>SQL (inherited from TCustomDADataset)</td>
<td>Used to provide a SQL statement that a query component executes when its Open method is called.</td>
</tr>
<tr>
<td>SQLDelete (inherited from TCustomDADataset)</td>
<td>Used to specify a SQL statement that will be used</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>SLLock (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 TCustomMyDataSet)</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the 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>

Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FetchAll</td>
<td>Defines whether to request all records of the query from</td>
</tr>
</tbody>
</table>
5.14.1.16.2.1  FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class
TMyTable

Syntax

```delphi
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.

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.
5.14.1.16.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

TMyTable

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

• TMyStoredProc.LockMode
• TMyQuery.LockMode

5.14.1.16.2.3 OrderFields Property

Used to build ORDER BY clause of SQL statements.

Class

TMyTable

Syntax

property OrderFields: string;

Remarks

TMyTable uses the OrderFields property to build ORDER BY clause of SQL statements. To
set several field names to this property separate them with commas.

TMyTable is reopened when OrderFields is being changed.

See Also
- TMyTable

5.14.1.16.2.4 TableName Property

Used to specify the name of the database table this component encapsulates.

Class
TMyTable

Syntax

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.

5.14.1.17 TMyTableOptions Class

This class allows setting up the behaviour of the TMyTable class.

For a list of all members of this type, see TMyTableOptions members.

Unit
MyAccess

Syntax

TMyTableOptions = class(TMyDataSetOptions);
Inheritance Hierarchy

TDADataSetOptions
   TMyDataSetOptions
      TMyTableOptions

© 1997-2021 Devart. All Rights Reserved.

5.14.1.17.1 Members

**TMyTableOptions** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AutoPrepare</strong></td>
<td>(inherited from TMyDataSetOptions) Used to execute automatic TCustomDADataset.Prepare on a query execution.</td>
</tr>
<tr>
<td><strong>AutoRefresh</strong></td>
<td>(inherited from TMyDataSetOptions) Used to automatically refresh dataset every AutoRefreshInterval seconds.</td>
</tr>
<tr>
<td><strong>AutoRefreshInterval</strong></td>
<td>(inherited from TMyDataSetOptions) Used to define in what time interval in seconds the Refresh or TCustomMyDataSet.RefreshQuick method of a DataSet is called.</td>
</tr>
<tr>
<td><strong>BinaryAsString</strong></td>
<td>(inherited from TMyDataSetOptions) Used to specify a method of representation of the BINARY and VARBINARY fields.</td>
</tr>
<tr>
<td><strong>CacheCalcFields</strong></td>
<td>(inherited from TDADataSetOptions) Used to enable caching of the TField CALCULATED and TField LOOKUP fields.</td>
</tr>
<tr>
<td><strong>CheckRowVersion</strong></td>
<td>(inherited from TMyDataSetOptions) Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or delete data.</td>
</tr>
<tr>
<td><strong>CompressBlobMode</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to store values of the BLOB fields in compressed form.</td>
</tr>
<tr>
<td><strong>CreateConnection</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>Used to specify if an additional connection to a server should be established to execute an additional query in the <strong>TCustomMyDataSet.FetchAll</strong> mode.</td>
</tr>
<tr>
<td><strong>DefaultValues</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>Used to fill the DefaultExpression property of TField objects with appropriate value.</td>
</tr>
<tr>
<td><strong>DetailDelay</strong> (inherited from <strong>TDADatasetOptions</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>EnableBoolean</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>Used to specify the method of representation of the TINYINT(1) fields.</td>
</tr>
<tr>
<td><strong>FieldsAsString</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>Used to store all non-BLOB fields as string (native MySQL format).</td>
</tr>
<tr>
<td><strong>FlatBuffers</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to control how a dataset treats data of the ftString and ftVarBytes fields.</td>
</tr>
<tr>
<td><strong>FullRefresh</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>Used to specify the fields to include in automatically generated SQL statement when calling the <strong>TCustomDADataSet.RefreshRecord</strong> method. Default value is false.</td>
</tr>
<tr>
<td><strong>HandlerIndex</strong></td>
<td>Used to assign an index and a value that this index should satisfy.</td>
</tr>
<tr>
<td><strong>InsertAllSetFields</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used to include all set dataset fields in the generated INSERT statement.</td>
</tr>
<tr>
<td><strong>LocalMasterDetail</strong> (inherited from <strong>TDADatasetOptions</strong>)</td>
<td>Used for <strong>TCustomDADataSet</strong> to use</td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>Local Filtering</td>
<td></td>
</tr>
<tr>
<td>LongStrings</td>
<td>TDADatasetOptions</td>
</tr>
<tr>
<td>MasterFieldsNullable</td>
<td>TDADatasetOptions</td>
</tr>
<tr>
<td>NullForZeroDate</td>
<td>TMyDataSetOptions</td>
</tr>
<tr>
<td>NumberRange</td>
<td>TMyDataSetOptions</td>
</tr>
<tr>
<td>QueryRecCount</td>
<td>TMyDataSetOptions</td>
</tr>
<tr>
<td>QuoteNames</td>
<td>TMyDataSetOptions</td>
</tr>
<tr>
<td>RemoveOnRefresh</td>
<td>TMyDataSetOptions</td>
</tr>
<tr>
<td>RequiredFields</td>
<td>TMyDataSetOptions</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>ReturnParams</code></td>
<td>the Required property of TField objects for NOT NULL fields.</td>
</tr>
<tr>
<td><code>SaveBlobsAsHex</code></td>
<td>Used to return the new value of the fields to dataset after insert or update.</td>
</tr>
<tr>
<td><code>SetEmptyStrToNull</code></td>
<td>Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML method.</td>
</tr>
<tr>
<td><code>SetFieldsReadOnly</code></td>
<td>Force replace of empty strings with NULL values in data. The default value is False.</td>
</tr>
<tr>
<td><code>StrictUpdate</code></td>
<td>Used for TCustomDADatabase to raise an exception when the number of updated or deleted records does not equal 1.</td>
</tr>
<tr>
<td><code>TrimFixedChar</code></td>
<td>Used to specify whether to discard all trailing spaces in string fields of the dataset.</td>
</tr>
<tr>
<td><code>TrimVarChar</code></td>
<td>Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.</td>
</tr>
<tr>
<td><code>UpdateAllFields</code></td>
<td>Used to include all dataset fields in the generated UPDATE and INSERT statements.</td>
</tr>
<tr>
<td><code>UpdateBatchSize</code></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><code>UseHandler</code></td>
<td>Used for the HANDLER statement to be used instead of the SELECT statement.</td>
</tr>
</tbody>
</table>
Properties of the `TMyTableOptions` class.

For a complete list of the `TMyTableOptions` class members, see the [TMyTableOptions Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>DetailDelay</td>
<td>Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master 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>
</tbody>
</table>
### MasterFieldsNullable (inherited from TDADataSetOptions)

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).

### 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>AutoPrepare (inherited from TMyDataSetOptions)</td>
<td>Used to execute automatic TCustomDADataset.Prepare on a query execution.</td>
</tr>
<tr>
<td>AutoRefresh (inherited from TMyDataSetOptions)</td>
<td>Used to automatically refresh dataset every AutoRefreshInterval seconds.</td>
</tr>
<tr>
<td>AutoRefreshInterval (inherited from TMyDataSetOptions)</td>
<td>Used to define in what time interval in seconds the Refresh or TCustomMyDataSet.RefreshQuick method of a DataSet is called.</td>
</tr>
<tr>
<td>BinaryAsString (inherited from TMyDataSetOptions)</td>
<td>Used to specify a method of representation of the BINARY and VARBINARY fields.</td>
</tr>
<tr>
<td>CheckRowVersion (inherited from TMyDataSetOptions)</td>
<td>Used to determine whether a dataset checks for rows modifications made by another user on automatic generation of SQL statement for update or</td>
</tr>
<tr>
<td><strong>CreateConnection</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>delete data. Used to specify if an additional connection to a server should be established to execute an additional query in the <code>TCustomMyDataSet.FetchAll=True</code> mode.</td>
</tr>
<tr>
<td><strong>DefaultValues</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>Used to fill the DefaultExpression property of TField objects with appropriate value.</td>
</tr>
<tr>
<td><strong>EnableBoolean</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>Used to specify the method of representation of the TINYINT(1) fields.</td>
</tr>
<tr>
<td><strong>FieldsAsString</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>Used to store all non-BLOB fields as string (native MySQL format).</td>
</tr>
<tr>
<td><strong>FullRefresh</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>Used to specify the fields to include in automatically generated SQL statement when calling the <code>TCustomDADataset.RefreshRecord</code> method. Default value is false.</td>
</tr>
<tr>
<td><strong>HandlerIndex</strong></td>
<td>Used to assign an index and a value that this index should satisfy.</td>
</tr>
<tr>
<td><strong>NullForZeroDate</strong> (inherited from <strong>TMyDataSetOptions</strong>)</td>
<td>Used for MySQL server to represent the value for for datetime fields with invalid values as Null or '0001-01-01' ('0100-01-01' for CLR).</td>
</tr>
<tr>
<td><strong>NumberRange</strong> (inherited from <strong>TMyDataSetOptions</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>TMyDataSetOptions</strong>)</td>
<td>Used for TCustomDADataset to perform additional query to get record count for this <code>SELECT</code> so the RecordCount property reflects the actual number of</td>
</tr>
<tr>
<td>Field Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>QuoteNames (inherited from TMyDataSetOptions)</td>
<td>Used for TCustomMyDataSet to quote all field names in autogenerated SQL statements.</td>
</tr>
<tr>
<td>RemoveOnRefresh (inherited from TMyDataSetOptions)</td>
<td>Used for dataset to remove a record locally if the RefreshRecord procedure can't find necessary record on the server.</td>
</tr>
<tr>
<td>RequiredFields (inherited from TMyDataSetOptions)</td>
<td>Used for TCustomDADataSet to set the Required property of TField objects for NOT NULL fields.</td>
</tr>
<tr>
<td>ReturnParams (inherited from TMyDataSetOptions)</td>
<td>Used to return the new value of the fields to dataset after insert or update.</td>
</tr>
<tr>
<td>SaveBlobsAsHex (inherited from TMyDataSetOptions)</td>
<td>Used to save BLOB data in hexadecimal format in a database after calling the TMyQuery.SaveToXML method.</td>
</tr>
<tr>
<td>SetEmptyStrToNull (inherited from TMyDataSetOptions)</td>
<td>Force replace of empty strings with NULL values in data. The default value is False.</td>
</tr>
<tr>
<td>SetFieldsReadOnly (inherited from TMyDataSetOptions)</td>
<td>Used to specify whether fields not belonging to the current updating table get read-only attribute.</td>
</tr>
<tr>
<td>StrictUpdate (inherited from TMyDataSetOptions)</td>
<td>Used for TCustomDADataSet to raise an exception when the number of updated or deleted records does not equal 1.</td>
</tr>
<tr>
<td>TrimFixedChar (inherited from TMyDataSetOptions)</td>
<td>Used to specify whether to discard all trailing spaces in string fields of the dataset.</td>
</tr>
<tr>
<td>TrimVarChar (inherited from TMyDataSetOptions)</td>
<td>Used to specify whether to discard all trailing spaces in the variable-length string fields of a dataset.</td>
</tr>
</tbody>
</table>
UseHandler

<table>
<thead>
<tr>
<th>UseHandler</th>
<th>Used for the HANDLER statement to be used instead of the SELECT statement.</th>
</tr>
</thead>
</table>

See Also
- **TMyTableOptions Class**
- **TMyTableOptions Class Members**

## 5.14.1.17.2.1 HandlerIndex Property

Used to assign an index and a value that this index should satisfy.

**Class**

**TMyTableOptions**

**Syntax**

```property```

HandlerIndex: `string`;

```endproperty```

**Remarks**

Use the HandlerIndex property to assign an index and a value that this index should satisfy.

## 5.14.1.17.2.2 UseHandler Property

Used for the HANDLER statement to be used instead of the SELECT statement.

**Class**

**TMyTableOptions**

**Syntax**

```property```

UseHandler: `boolean` default `False`;

```endproperty```

**Remarks**


If this option is enabled, the HANDLER statement is used instead of the SELECT statement.

From the MySQL Reference Manual:

"There are several reasons to use the HANDLER interface instead of normal SELECT statements:

HANDLER is faster than SELECT:

A designated storage engine handler object is allocated for the HANDLER ... OPEN. The object is reused for subsequent HANDLER statements for that table; it need not be reinitialized for each one.

There is less parsing involved.

There is no optimizer or query-checking overhead.

The table does not have to be locked between two handler requests.

The handler interface does not have to provide a consistent look of the data (for example, dirty reads are allowed), so the storage engine can use optimizations that SELECT does not normally allow.

For applications that use a low-level ISAM-like interface, HANDLER makes it much easier to port them to MySQL.

HANDLER enables you to traverse a database in a manner that is difficult (or even impossible) to accomplish with SELECT. The HANDLER interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database."

The FilterSQL property is used to assign the WHERE condition. To return a specific number of rows, assign the Limit property.

If the Limit property equals -1 and the UseHandler option is set to True, requested records count equals to MaxInt.

The default value of the UseHandler option is False.

5.14.1.18 TMyTransaction Class

A component for managing transactions.

For a list of all members of this type, see TMyTransaction members.
Unit
MyAccess

Syntax

```
TMyTransaction = class(TDATransaction);
```

Remarks
The TMyTransaction component is used to manage transactions in your application.

Inheritance Hierarchy

TDATransaction
TMyTransaction

© 1997-2021
Devart. All Rights Reserved.

5.14.1.18.1 Members

**TMyTransaction** 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>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit (inherited from TDATransaction)</td>
<td>Commits the current transaction.</td>
</tr>
<tr>
<td>Rollback (inherited from TDATransaction)</td>
<td>Discards all modifications of data associated with the current transaction and ends</td>
</tr>
</tbody>
</table>
### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnCommit</strong> (inherited from TDATransaction)</td>
<td>Occurs after the transaction has been successfully committed.</td>
</tr>
<tr>
<td><strong>OnCommitRetaining</strong> (inherited from TDATransaction)</td>
<td>Occurs after CommitRetaining has been executed.</td>
</tr>
<tr>
<td><strong>OnError</strong> (inherited from TDATransaction)</td>
<td>Used to process errors that occur during executing a transaction.</td>
</tr>
<tr>
<td><strong>OnRollback</strong> (inherited from TDATransaction)</td>
<td>Occurs after the transaction has been successfully rolled back.</td>
</tr>
<tr>
<td><strong>OnRollbackRetaining</strong> (inherited from TDATransaction)</td>
<td>Occurs after RollbackRetaining has been executed.</td>
</tr>
</tbody>
</table>

### 5.14.1.19 TMyUpdateSQL Class

A component for tuning update operations for the DataSet component.

For a list of all members of this type, see [TMyUpdateSQL members](#).

#### Unit

**MyAccess**

#### Syntax

```plaintext
TMyUpdateSQL = class(TCustomDAUpdateSQL);
```

#### Remarks

Use the TMyUpdateSQL component to provide DML statements for the dataset components that return read-only result set. This component also allows setting objects that can be used...
for executing update operations. You may prefer to use directly SQLInsert, SQLUpdate, and SQLDelete properties of the TCustomDADataSet descendants.

Inheritance Hierarchy

TCustomDAUpdateSQL
   TMyUpdateSQL

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataSet</strong></td>
<td>(inherited from TCustomDAUpdateSQL) Used to hold a reference to the TCustomDADataSet object that is being updated.</td>
</tr>
<tr>
<td><strong>DeleteObject</strong></td>
<td>(inherited from TCustomDAUpdateSQL) Provides ability to perform advanced adjustment of the delete operations.</td>
</tr>
<tr>
<td><strong>DeleteSQL</strong></td>
<td>(inherited from TCustomDAUpdateSQL) Used when deleting a record.</td>
</tr>
<tr>
<td><strong>InsertObject</strong></td>
<td>(inherited from TCustomDAUpdateSQL) Provides ability to perform advanced adjustment of insert operations.</td>
</tr>
<tr>
<td><strong>InsertSQL</strong></td>
<td>(inherited from TCustomDAUpdateSQL) Used when inserting a record.</td>
</tr>
<tr>
<td><strong>LockObject</strong></td>
<td>(inherited from TCustomDAUpdateSQL) Provides ability to perform advanced adjustment of lock operations.</td>
</tr>
<tr>
<td><strong>LockSQL</strong></td>
<td>(inherited from TCustomDAUpdateSQL) Used to lock the current record.</td>
</tr>
<tr>
<td><strong>ModifyObject</strong></td>
<td>(inherited from TCustomDAUpdateSQL) Provides ability to perform advanced adjustment of modify operations.</td>
</tr>
<tr>
<td><strong>ModifySQL</strong></td>
<td>(inherited from TCustomDAUpdateSQL) Used when updating a record.</td>
</tr>
<tr>
<td><strong>RefreshObject</strong></td>
<td>(inherited from TCustomDAUpdateSQL) Provides ability to perform advanced adjustment of refresh operations.</td>
</tr>
</tbody>
</table>
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>Apply (inherited from TCustomDAUpdateSQL)</td>
<td>Sets parameters for a SQL statement and executes it to update a record.</td>
</tr>
<tr>
<td>ExecSQL (inherited from TCustomDAUpdateSQL)</td>
<td>Executes a SQL statement.</td>
</tr>
</tbody>
</table>

5.14.2 Types

Types in the MyAccess unit.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyUpdateExecuteEvent</td>
<td>This type is used for the E:Devart.MyDac.TCustomMyDataSet.AfterUpdateExecute and E:Devart.MyDac.TCustomMyDataSet.BeforeUpdateExecute events.</td>
</tr>
</tbody>
</table>
5.14.2.1 **TMyUpdateExecuteEvent Procedure Reference**

This type is used for the E:Devart.MyDac.TCustomMyDataSet.AfterUpdateExecute and E:Devart.MyDac.TCustomMyDataSet.BeforeUpdateExecute events.

**Unit**

*MyAccess*

**Syntax**

```pascal
TMyUpdateExecuteEvent = procedure (Sender: TCustomMyDataSet; 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.

5.14.3 **Enumerations**

Enumerations in the *MyAccess* unit.

**Enumerations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TLockRecordType</strong></td>
<td>Specifies the type of the record locking.</td>
</tr>
<tr>
<td><strong>TLockType</strong></td>
<td>Specifies the type of the table locking.</td>
</tr>
<tr>
<td><strong>TMyIsolationLevel</strong></td>
<td>Specifies the extent to which all outside transactions interfere with subsequent transactions of current connection.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
5.14.3.1 TLockRecordType Enumeration

Specifies the type of the record locking.

Unit
MySQLAccess

Syntax

TLockRecordType = (lrImmediately, lrDelayed);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrDelayed</td>
<td>Locking is performed just at the time of execution Post for this record.</td>
</tr>
<tr>
<td>lrImmediately</td>
<td>Checks for locking directly on calling the Lock method. In this case if set Lock(lrImmediately) call to BeforeEdit event you can unable the same row in the table to be modified by several users at the same time.</td>
</tr>
</tbody>
</table>

5.14.3.2 TLockType Enumeration

Specifies the type of the table locking.

Unit
MySQLAccess

Syntax

TLockType = (ltRead, ltReadLocal, ltWrite, ltWriteLowPriority);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ltRead</td>
<td>All connections including the current one can read only from the specified table.</td>
</tr>
<tr>
<td>ltReadLocal</td>
<td>Similar to ltRead. Moreover, allows non-conflicting INSERT statements to execute while the lock is held.</td>
</tr>
<tr>
<td>ltWrite</td>
<td>The current connection can read and write to the table. Other</td>
</tr>
</tbody>
</table>
connections wait for the UnlockTable call.

**ltWriteLowPriority**
You can use ltWriteLowPriority locks to allow other threads to obtain ltRead locks while the thread is waiting for the ltWrite lock. You should use only ltWriteLowPriority locks if you are sure that eventually there will be a time when no threads will have a ltRead lock.

---

### 5.14.3.3 TMyIsolationLevel Enumeration

Specifies the extent to which all outside transactions interfere with subsequent transactions of current connection.

**Unit**
MyAccess

**Syntax**

```
TMyIsolationLevel = (ilReadCommitted, ilReadUnCommitted, ilRepeatableRead, ilSerializable);
```

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ilReadCommitted</td>
<td>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 isolation level. The default value.</td>
</tr>
<tr>
<td>ilReadUnCommitted</td>
<td>The most unrestricted level of transaction isolation. All types of data access interferences are possible. Mainly used for browsing database and to receive instant data with prospective changes.</td>
</tr>
<tr>
<td>ilRepeatableRead</td>
<td>Prevents concurrent transactions from modifying data in 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).</td>
</tr>
<tr>
<td>ilSerializable</td>
<td>The most restricted level of transaction isolation. Database server isolates data involved in current transaction by putting additional processing on range locks. Used to put aside all undesired effects observed in concurrent accesses to the same set of data, but may lead to a greater latency at times of</td>
</tr>
</tbody>
</table>
5.14.4 Routines

Routines in the MyAccess unit.

Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetServerList</td>
<td>Returns the list of the MySQL servers in LAN. MySQL server does not provide usual ways of such list getting, so it can be incomplete.</td>
</tr>
</tbody>
</table>

5.14.4.1 GetServerList Procedure

Returns the list of the MySQL servers in LAN. MySQL server does not provide usual ways of such list getting, so it can be incomplete.

Unit

MyAccess

Syntax

```pascal
procedure GetServerList(List: TStrings);
```

Parameters

List

the list of the MySQL servers in LAN.
5.14.5 Constants

Constants in the MyAccess unit.

### Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyDACVersion</td>
<td>Read this constant to get current version number for MyDAC.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.14.5.1 MyDACVersion Constant

Read this constant to get current version number for MyDAC.

**Unit**

MyAccess

**Syntax**

```
MyDACVersion = '10.4.1';
```

© 1997-2021 Devart. All Rights Reserved.

5.15 MyBackup

This unit contains implementation of the TMyBackup component.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyBackup</td>
<td>A component that serves for backup copying specified tables on the server.</td>
</tr>
</tbody>
</table>

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyBackupMode</td>
<td>Specifies the mode of TMyBackup work.</td>
</tr>
<tr>
<td>TMyBackupPriority</td>
<td>Specifies priority of the TMyBackup.Restore operation.</td>
</tr>
<tr>
<td>TMyRestoreDuplicates</td>
<td>Specifies the behaviour on detection records with repeated key fields on the execution of the TMyBackup.Restore method.</td>
</tr>
</tbody>
</table>

### Classes

#### Classes in the **MyBackup** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyBackup</td>
<td>A component that serves for backup copying specified tables on the server.</td>
</tr>
</tbody>
</table>

A component that serves for backup copying specified tables on the server.

For a list of all members of this type, see **TMyBackup** members.
**MyBackup**

**Syntax**

```delphi
TMyBackup = class(TComponent);
```

**Remarks**

Serves for backup copying specified tables on the server. Supports working in two modes defined by the `TMyBackup.Mode` property - bmText and bmBinary.

The list of tables is specified in `TMyBackup.TableNames`.

Use the `TMyBackup.Path` property to specify the path to the server.

TMyBackup works on the server side that greatly affects on performance. However, it has some restrictions. Firstly, backup is performed only for the tables, database structure and user rights that are not stored. Secondly, files created on the server cannot be modified and deleted by MySQL tools.

**Note:** When using bmBinary mode, both servers must have the same format of the tables (must be the same version). About compatibility of formats storing data please see MySQL Reference Manual.

**See Also**

- `TMyBackup.Backup`
- `TMyBackup.Restore`
- `TMyDump`

© 1997-2021 Devart. All Rights Reserved.

---

5.15.1.1.1 **Members**

**TMyBackup** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackupPriority</td>
<td>Specifies priority of the <code>TMyBackup.Restore</code> operation.</td>
</tr>
<tr>
<td>Connection</td>
<td>Used to specify a connection object that will be</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display the statement being executed.</td>
</tr>
<tr>
<td>Duplicates</td>
<td>Used to specify the behaviour on detection records with repeated key fields on the execution of the TMyBackup.Restore method.</td>
</tr>
<tr>
<td>EnclosedBy</td>
<td>Used to specify a string (a character set) with which a field can be quoted.</td>
</tr>
<tr>
<td>EscapedBy</td>
<td>Used to specify a symbol that will be used as a special symbol pointer.</td>
</tr>
<tr>
<td>Fields</td>
<td>Holds the list of the fields that will be saved.</td>
</tr>
<tr>
<td>FieldsTerminatedBy</td>
<td>Used to specify field delimiter.</td>
</tr>
<tr>
<td>IgnoreLines</td>
<td>Determines if the specified rows number at the beginning of the file will be ignored or not.</td>
</tr>
<tr>
<td>LinesTerminatedBy</td>
<td>Used to specify row delimiter.</td>
</tr>
<tr>
<td>Local</td>
<td>Used to specify that TMyBackup.Path is a local path but not a path on the server.</td>
</tr>
<tr>
<td>Mode</td>
<td>Used to control modes of TMyBackup work.</td>
</tr>
<tr>
<td>Path</td>
<td>Holds a path on the server to the folder where data files will be stored.</td>
</tr>
<tr>
<td>TableNames</td>
<td>Holds the list of tables which will be used in the script.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup</td>
<td>Copies current tables by the specified path.</td>
</tr>
</tbody>
</table>
### Restore

Restores tables.

### Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnTableMsg</td>
<td>Occurs on executing of the TMyBackup.Restore operation.</td>
</tr>
</tbody>
</table>

5.15.1.2 Properties

Properties of the TMyBackup class.

For a complete list of the TMyBackup class members, see the TMyBackup Members topic.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackupPriority</td>
<td>Specifies priority of the TMyBackup.Restore operation.</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 being executed.</td>
</tr>
<tr>
<td>Duplicates</td>
<td>Used to specify the behaviour on detection records with repeated key fields on the execution of the TMyBackup.Restore method.</td>
</tr>
<tr>
<td>EnclosedBy</td>
<td>Used to specify a string (a character set) with which a field can be quoted.</td>
</tr>
<tr>
<td>EscapedBy</td>
<td>Used to specify a symbol that will be used as a special symbol pointer.</td>
</tr>
<tr>
<td>Fields</td>
<td>Holds the list of the fields</td>
</tr>
</tbody>
</table>
### 5.15.1.1.2.1 BackupPriority Property

Specifies priority of the **Restore** operation.

#### Class

**TMyBackup**

#### Syntax

```plaintext
property BackupPriority: TMyBackupPriority default bpDefault;
```

#### Remarks

Specifies a priority on **Restore** operation.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FieldsTerminatedBy</strong></td>
<td>Used to specify field delimiter.</td>
</tr>
<tr>
<td><strong>IgnoreLines</strong></td>
<td>Determines if the specified rows number at the beginning of the file will be ignored or not.</td>
</tr>
<tr>
<td><strong>LinesTerminatedBy</strong></td>
<td>Used to specify row delimiter.</td>
</tr>
<tr>
<td><strong>Local</strong></td>
<td>Used to specify that <strong>TMyBackup.Path</strong> is a local path but not a path on the server.</td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td>Used to control modes of TMyBackup work.</td>
</tr>
<tr>
<td><strong>Path</strong></td>
<td>Holds a path on the server to the folder where data files will be stored.</td>
</tr>
<tr>
<td><strong>TableNames</strong></td>
<td>Holds the list of tables which will be used in the script.</td>
</tr>
</tbody>
</table>
5.15.1.1.2.2  Connection Property

Used to specify a connection object that will be used to connect to a data store.

Class

TMyBackup

Syntax

```property
connection: TCustomMyConnection;
```

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 TCustomMyConnection objects.

At run-time, set the Connection property to reference an existing TCustomMyConnection object.

See Also

• TCustomMyConnection

5.15.1.1.2.3  Debug Property

Used to display the statement being executed.

Class

TMyBackup
Syntax

```plaintext
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement being executed.

The default value is False.

**Note:** To enable debug form display you should explicitly include MyDacVcl unit in your project.

See Also

- `TCustomDASQL.Debug`
- `TCustomDADataSet.Debug`

5.15.1.1.2.4 Duplicates Property

Used to specify the behaviour on detection records with repeated key fields on the execution of the `Restore` method.

Class

`TMyBackup`

Syntax

```plaintext
property Duplicates: TMyRestoreDuplicates default bdError;
```

Remarks

Use the Duplicates property to specify the behaviour on detection records with repeated key fields on the execution of the `Restore` method.

Used only if `Mode` is `bmText`.

See Also

- `Mode`
- `Restore`
5.15.1.1.2.5 EnclosedBy Property

Used to specify a string (a character set) with which a field can be quoted.

Class

TMyBackup

Syntax

```delphi
property EnclosedBy: string;
```

Remarks

Use the EnclosedBy property to specify a string (a character set) with which a field can be quoted. By default it is an empty string.

Used only if `Mode` is `bmText`.

See Also

- Backup
- Mode
- EscapedBy
- FieldsTerminatedBy
- LinesTerminatedBy

5.15.1.1.2.6 EscapedBy Property

Used to specify a symbol that will be used as a special symbol pointer.

Class

TMyBackup

Syntax

```delphi
property EscapedBy: string;
```

Remarks
Use the EscapedBy property to specify a symbol that will be used as a special symbol pointer to define the entering of a special symbol. By default such pointer is `'\'`.

Used only if \texttt{Mode} is \texttt{bmText}.

\subsection*{See Also}
- \texttt{Backup}
- \texttt{Mode}
- \texttt{EnclosedBy}
- \texttt{FieldsTerminatedBy}
- \texttt{LinesTerminatedBy}

\section*{Fields Property}

Holds the list of the fields that will be saved.

\textbf{Class}

\texttt{TMyBackup}

\textbf{Syntax}

\begin{verbatim}
property Fields: string;
\end{verbatim}

\textbf{Remarks}

The Fields property is used to contain the list of the fields that will be saved. If not specified, all fields will be stored.

Use this property on execution \texttt{Backup} for a single table (see \texttt{TableNames}).

Used only if \texttt{Mode} is \texttt{bmText}.

\subsection*{See Also}
- \texttt{Mode}
- \texttt{TableNames}
- \texttt{Backup}
- \texttt{Restore}
5.15.1.2.8 FieldsTerminatedBy Property

Used to specify field delimiter.

Class
TMyBackup

Syntax

```
property FieldsTerminatedBy: string;
```

Remarks

Use the FieldsTerminatedBy property to specify a string that will be used as field delimiter. By default field delimiter is '\t'.

Used only if Mode is bmText.

See Also
- Backup
- Mode
- EnclosedBy
- EscapedBy
- LinesTerminatedBy

5.15.1.2.9 IgnoreLines Property

Determines if the specified rows number at the beginning of the file will be ignored or not.

Class
TMyBackup

Syntax

```
property IgnoreLines: integer default 0;
```

Remarks
Use the IgnoreLines property to determine if the specified rows number at the beginning of the file will be ignored or not.

Use this property if you need:
• create non-typical formatted text file to be processed by the third-party applications
• fast load of big volume data to the database generated by the third-party application.

Used only if Mode is bmText.

See Also
• Mode
• Restore

Class
TMyBackup

Syntax

| property LinesTerminatedBy: string; |

Remarks

Use the LinesTerminatedBy property to specify a string that will be used as row delimiter. By default row delimiter is '\n'.

Used only if Mode is bmText.

See Also
• Backup
• Mode
• EnclosedBy
• EscapedBy
• FieldsTerminatedBy
5.15.1.1.2.11 Local Property

Used to specify that Path is a local path but not a path on the server.

Class

TMyBackup

Syntax

```plaintext
property Local: boolean default False;
```

Remarks

Use the Local property to specify that Path is a local path but not a path on the server on the execution of the Restore method.

Used only if Mode is bmText.

See Also

- Mode
- Restore

5.15.1.1.2.12 Mode Property

Used to control modes of TMyBackup work.

Class

TMyBackup

Syntax

```plaintext
property Mode: TMyBackupMode default bmBinary;
```

Remarks

Use the Mode property to control modes of TMyBackup work.

**Note:** By security reasons MySQL does not allow to overwrite files that already exist. Also MySQL requires a path to the server to be created beforehand.
When using bmBinary mode, both servers must have the same format of the tables (must be the same version). About the compatibility of formats storing data please see MySQL Reference Manual.

See Also
• Backup
• Restore

5.15.1.1.2.13 Path Property

Holds a path on the server to the folder where data files will be stored.

Class
TMyBackup

Syntax

property Path: string;

Remarks

Use the Path property to contain a path on the server to the folder where data files will be stored. Path must exist before executing the Backup and Restore operations.

See Also
• Backup
• Restore

5.15.1.1.2.14 TableNames Property

Holds the list of tables which will be used in the script.

Class
TMyBackup
Syntax

```pascal
property TableNames: string;
```

Remarks

Use the TableNames property to hold the list of tables which will be used in the script. Table names are separated by comma or semicolon. If it has an empty value, all the tables presented in the database will be processed.

See Also

- [Backup](#)
- [Restore](#)
- [TCustomDACConnection.GetTableNames](#)

© 1997-2021

Devart. All Rights Reserved.

5.15.1.1.3 Methods

Methods of the **TMyBackup** class.

For a complete list of the **TMyBackup** class members, see the [TMyBackup Members](#) topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup</td>
<td>Copies current tables by the specified path.</td>
</tr>
<tr>
<td>Restore</td>
<td>Restores tables.</td>
</tr>
</tbody>
</table>

See Also

- [TMyBackup Class](#)
- [TMyBackup Class Members](#)

© 1997-2021

Devart. All Rights Reserved.

Request Support   DAC Forum   Provide Feedback
5.15.1.1.3.1 Backup Method

Copies current tables by the specified path.

Class
TMyBackup

Syntax

```plaintext
procedure Backup;
```

Remarks

Call the Backup method to copy current tables by the specified path (see Path property).

A list of tables is specified in the TableNames property. If the list isn't specified, all tables will be copied.

Use the Mode property to specify a copy mode.

If Mode is bmBinary before backup copying of table files, tables are TCustomMyDataSet.LockTable on writing and TMyServerControl.Flush is called.

See Also
- Mode
- TableNames
- Path

5.15.1.1.3.2 Restore Method

Restores tables.

Class
TMyBackup

Syntax

```plaintext
procedure Restore;
```

Remarks
Call the Restore method to restore tables.

See Also
- `OnTableMsg`

For a complete list of the `TMyBackup` class members, see the `TMyBackup Members` topic.

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OnTableMsg</code></td>
<td>Occurs on executing of the <code>TMyBackup.Restore</code> operation.</td>
</tr>
</tbody>
</table>

### See Also
- `TMyBackup Class`
- `TMyBackup Class Members`

5.15.1.1.4  Events

Events of the `TMyBackup` class.

### OnTableMsg Event

Occurs on executing of the `Restore` operation.

**Class**

`TMyBackup`

**Syntax**

```pascal
property OnTableMsg: TMyTableMsgEvent;
```

**Remarks**

The `OnTableMsg` event occurs on executing of the `Restore` operation.
5.15.2 Types

Types in the MyBackup unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyTableMsgEvent</td>
<td>This type is used for the TMyBackup.OnTableMsg event.</td>
</tr>
</tbody>
</table>

5.15.2.1 TMyTableMsgEvent Procedure Reference

This type is used for the TMyBackup.OnTableMsg event.

Unit

MyBackup

Syntax

TMyTableMsgEvent = procedure (Sender: TObject; TableName: string; MsgText: string) of object;

Parameters

Sender
An object that raised the event.

TableName
Holds the table name.

MsgText
Holds the message text.

© 1997-2021
Devart. All Rights Reserved.
5.15.3 Enumerations

Enumerations in the MyBackup unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyBackupMode</td>
<td>Specifies the mode of TMyBackup work.</td>
</tr>
<tr>
<td>TMyBackupPriority</td>
<td>Specifies priority of the TMyBackup.Restore operation.</td>
</tr>
<tr>
<td>TMyRestoreDuplicates</td>
<td>Specifies the behaviour on detection records with repeated key fields on the execution of the TMyBackup.Restore method.</td>
</tr>
</tbody>
</table>

5.15.3.1 TMyBackupMode Enumeration

Specifies the mode of TMyBackup work.

Unit
MyBackup

Syntax

TMyBackupMode = (bmBinary, bmText);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmBinary</td>
<td>The fastest way of saving data. Files with the tables are copied physically to the specified folder (Path). This mode is not supported for several table types, for example, InnoDB.</td>
</tr>
<tr>
<td>bmText</td>
<td>Data from the tables is copied as a text file. To specify its format use TMyBackup.EnclosedBy, TMyBackup.EscapedBy, TMyBackup.FieldsTerminatedBy, TMyBackup.LinesTerminatedBy properties. This mode can be used for all table types.</td>
</tr>
</tbody>
</table>
5.15.3.2 TMyBackupPriority Enumeration

Specifies priority of the TMyBackup.Restore operation.

Unit
MyBackup

Syntax
TMyBackupPriority = (bpDefault, bpLowPriority, bpConcurrent);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bpConcurrent</td>
<td>Restore will be executed simultaneously with queries from other connections.</td>
</tr>
<tr>
<td>bpDefault</td>
<td>Other connections wait for finishing Restore.</td>
</tr>
<tr>
<td>bpLowPriority</td>
<td>Execution of Restore will be suspended until other connections stop reading from the table.</td>
</tr>
</tbody>
</table>

5.15.3.3 TMyRestoreDuplicates Enumeration

Specifies the behaviour on detection records with repeated key fields on the execution of the TMyBackup.Restore method.

Unit
MyBackup

Syntax
TMyRestoreDuplicates = (bdIgnore, bdReplace, bdError);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bdError</td>
<td>Generate an error, ignore the rest of the file.</td>
</tr>
</tbody>
</table>
5.16 **MyBuilderClient**

This unit contains implementation of the MyBuilder class.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyBuilder</td>
<td>Serves to manage SQL Builder for MySQL Add-in.</td>
</tr>
</tbody>
</table>

5.16.1 **Classes**

Classes in the `MyBuilderClient` unit.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyBuilder</td>
<td>Serves to manage SQL Builder for MySQL Add-in.</td>
</tr>
</tbody>
</table>

5.16.1.1 **TMyBuilder Class**

Serves to manage SQL Builder for MySQL Add-in.

For a list of all members of this type, see `TMyBuilder` members.

**Unit**

`MyBuilderClient`

**Syntax**
TMyBuilder = class(TComponent);

5.16.1.1 Members

**TMyBuilder** class overview.

5.17 **MyClasses**

This unit contains implementation of the EMyError class.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EMyError</strong></td>
<td>A base class that is raised when MySQL server returns error as a result.</td>
</tr>
</tbody>
</table>

### Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TMyProtocol</strong></td>
<td>Specifies which protocol to use when connecting to server.</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Strings65535ToMemo</em></td>
<td>Control flow functions of MySQL (like IF, CASE) change data type of LONGMEMO and LONGBLOB fields.</td>
</tr>
</tbody>
</table>
5.17.1 Classes

Classes in the **MyClasses** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMyError</td>
<td>A base class that is raised when MySQL server returns error as a result.</td>
</tr>
</tbody>
</table>

5.17.1.1 EMyError Class

A base class that is raised when MySQL server returns error as a result.

For a list of all members of this type, see [EMyError members](#).

**Unit**

**MyClasses**

**Syntax**

```plaintext
EMyError = class (EDAError);
```

**Remarks**

EMyError is raised when MySQL Server returns error as a result, for example, of an attempt to execute invalid SQL statement. Use EMyError in an exception-handling block.

**Inheritance Hierarchy**

[EDAError](#)  
EMyError

**See Also**

- [EDAError](#)
5.17.1.1.1 Members

**EMyError** class overview.

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>
<tr>
<td>LineNumber</td>
<td>Contains the number of a query line that caused an error.</td>
</tr>
</tbody>
</table>

5.17.1.1.2 Properties

Properties of the **EMyError** class.

For a complete list of the **EMyError** class members, see the **EMyError Members** topic.

Public

<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>
<tr>
<td>LineNumber</td>
<td>Contains the number of a query line that caused an error.</td>
</tr>
</tbody>
</table>

See Also

- **EMyError Class**
- **EMyError Class Members**

© 1997-2021 Devart. All Rights Reserved.
5.17.1.2.1 LineNumber Property

Contains the number of a query line that caused an error.

Class

EMyError

Syntax

```pascal
property LineNumber: integer;
```

Remarks

If an error, having `EDAError.ErrorCode = ER_PARSE_ERROR`, occurred during query execution, LineNumber property contains the number of a query line that caused an error. MyDAC will retrieve this information from the error text automatically.

© 1997-2021
Devart. All Rights Reserved.

5.17.2 Enumerations

Enumerations in the `MyClasses` unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyProtocol</td>
<td>Specifies which protocol to use when connecting to server.</td>
</tr>
</tbody>
</table>

© 1997-2021
Devart. All Rights Reserved.

5.17.2.1 TMyProtocol Enumeration

Specifies which protocol to use when connecting to server.

Unit

`MyClasses`

Syntax
TMyProtocol = (mpDefault, mpTCP, mpPipe, mpMemory, mpSSL, mpHttp);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpDefault</td>
<td>Similar to mpTCP, except the cases when you connect to a local server and the OS supports sockets (Unix) or named pipes (Windows), they are used instead of TCP/IP to connect to the server.</td>
</tr>
<tr>
<td>mpHttp</td>
<td>Uses HTTP <a href="#">Network Tunneling</a> to connect to the server.</td>
</tr>
<tr>
<td>mpMemory</td>
<td>Uses SharedMem to connect to the server. Can be used with Direct set to False and libmysql.dll 4.1.</td>
</tr>
<tr>
<td>mpPipe</td>
<td>Uses NamedPipes to connect to the server.</td>
</tr>
<tr>
<td>mpSocket</td>
<td>Uses sockets to connect to the server. Can be used with Direct set to False and libmysql.dll 4.1.</td>
</tr>
<tr>
<td>mpSSL</td>
<td>Uses protected SSL connection with the server. To use SSL you need to set <a href="#">TMyConnection.SSLOptions</a>.</td>
</tr>
<tr>
<td>mpTCP</td>
<td>Uses TCP/IP to connect to the server.</td>
</tr>
</tbody>
</table>

5.17.3 Variables

Variables in the [MyClasses](#) unit.

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__Strings65535ToMemo</td>
<td>Control flow functions of MySQL (like IF, CASE) change data type of LONGMEMO and LONGBLOB fields.</td>
</tr>
</tbody>
</table>

5.17.3.1 __Strings65535ToMemo Variable

Control flow functions of MySQL (like IF, CASE) change data type of LONGMEMO and LONGBLOB fields.
Unit

**MyClasses**

Syntax

```
__Strings65535ToMemo: boolean = True;
```

Remarks

Control flow functions of MySQL (like IF, CASE) change data type of LONGMEMO and LONGBLOB fields. It causes wrong description of these fields by MyDAC and truncating their data. To avoid these problems, MyDAC tries to restore the correct data type. This behaviour was introduced in MyDAC 5.10.0.9. To disable this behaviour, set the __Strings65535ToMemo variable to False.

5.18 **MyConnectionPool**

This unit contains the TMyConnectionPoolManager class for managing connection pool.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyConnectionPoolManager</td>
<td>A class of methods that are used for managing MyDAC connection pool.</td>
</tr>
</tbody>
</table>

5.18.1 **Classes**

Classes in the **MyConnectionPool** unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyConnectionPoolManager</td>
<td>A class of methods that are used for managing MyDAC connection pool.</td>
</tr>
</tbody>
</table>
5.18.1.1  **TMyConnectionPoolManager Class**

A class of methods that are used for managing MyDAC connection pool.

For a list of all members of this type, see [TMyConnectionPoolManager](#) members.

**Unit**

`MyConnectionPool`

**Syntax**

```plaintext
TMyConnectionPoolManager = class(TCRConnectionPoolManager);
```

**Remarks**

Use the TMyConnectionPoolManager methods to manage MyDAC connection pool.

**Inheritance Hierarchy**

TCRConnectionPoolManager  
  TMyConnectionPoolManager

**See Also**

- [Connection Pooling](#)

---

5.19  **MyDacVcl**

This unit contains the visual constituent of MyDAC.

**Classes**
### 5.19.1 Classes

Classes in the **MyDacVcl** unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyConnectDialog</td>
<td>A class that provides a dialog box for user to supply his login information.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

#### 5.19.1.1 TMyConnectDialog Class

A class that provides a dialog box for user to supply his login information.

For a list of all members of this type, see [TMyConnectDialog](#) members.

**Unit**

**MyDacVcl**

**Syntax**

```pascal
TMyConnectDialog = class(TCustomConnectDialog);
```

**Remarks**

The TMyConnectDialog component is a direct descendant of TCustomConnectDialog class. Use TMyConnectDialog to provide dialog box for user to supply server name, user name, and password. You may want to customize appearance of dialog box using this class's properties.

**Inheritance Hierarchy**
**TCustomConnectDialog**

**TMyConnectDialog**

See Also
- **TCustomDAConnection.ConnectDialog**

© 1997-2021
Devar. All Rights Reserved.

## Members

### TMyConnectDialog 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>Connection</td>
<td>Holds the TMyConnection component which uses TMyConnectDialog 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>
</tbody>
</table>
**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>(inherited from <a href="#">TCustomConnectDialog</a>) 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>(inherited from <a href="#">TCustomConnectDialog</a>) Retrieves a list of available server names.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
[Request Support](#) DAC Forum [Provide Feedback](#)

5.19.1.2 Properties

Properties of the **TMyConnectDialog** class.

For a complete list of the **TMyConnectDialog** class members, see the [TMyConnectDialog Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CancelButton</strong></td>
<td>(inherited from <a href="#">TCustomConnectDialog</a>) Used to specify the label for the Cancel button.</td>
</tr>
<tr>
<td><strong>Caption</strong></td>
<td>(inherited from <a href="#">TCustomConnectDialog</a>) Used to set the caption of</td>
</tr>
<tr>
<td><strong>ConnectButton</strong> (inherited from <strong>TCustomConnectDialog</strong>)</td>
<td>dialog box.</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DatabaseLabel</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DialogClass</strong> (inherited from <strong>TCustomConnectDialog</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>LabelSet</strong> (inherited from <strong>TCustomConnectDialog</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>PasswordLabel</strong> (inherited from <strong>TCustomConnectDialog</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>PortLabel</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Retries</strong> (inherited from <strong>TCustomConnectDialog</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>SavePassword</strong> (inherited from <strong>TCustomConnectDialog</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>ServerLabel</strong> (inherited from <strong>TCustomConnectDialog</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>ShowDatabase</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ShowPort</strong></td>
<td></td>
</tr>
<tr>
<td><strong>StoreLogInfo</strong> (inherited from <strong>TCustomConnectDialog</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>UsernameLabel</strong> (inherited from <strong>TCustomConnectDialog</strong>)</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**
- **TMyConnectDialog Class**
5.19.1.1.2.1 Connection Property

Holds the TMyConnection component which uses TMyConnectDialog object.

Class

TMyConnectDialog

Syntax

```pascal
property Connection: TCustomMyConnection;
```

Remarks

Use the Connection property to learn which TMyConnection component uses TMyConnectDialog object. This property is read-only.

See Also

- TCustomDACConnection.ConnectDialog

5.19.1.1.2.2 DatabaseLabel Property

Used to specify a prompt for database edit.

Class

TMyConnectDialog

Syntax

```pascal
property DatabaseLabel: string;
```

Remarks

Use the DatabaseLabel property to specify a prompt for database edit.

See Also
5.19.1.2.3 PortLabel Property

Used to specify a prompt for port edit.

Class

TMyConnectDialog

Syntax

property PortLabel: string;

Remarks

Use the PortLabel property to specify a prompt for port edit.

See Also

• ShowPort

5.19.1.2.4 ShowDatabase Property

Used to display a field for entering database at connect dialog.

Class

TMyConnectDialog

Syntax

property ShowDatabase: boolean default True;

Remarks

Use the ShowDatabase property to display a field for entering database at connect dialog.

The default value is True.

See Also
5.19.1.2.5 Show Port Property

Used to display a field for entering port at connect dialog.

Class

TMyConnectDialog

Syntax

```
property ShowPort: boolean default True;
```

Remarks

Use the ShowPort property to display a field for entering port at connect dialog.

The default value is True.

See Also

- PortLabel
- ShowDatabase

5.20 MyDump

This unit contains implementation of the TMyDump component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyDump</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>TMyDumpOptions</td>
<td>This class allows setting up</td>
</tr>
</tbody>
</table>
the behaviour of the TMyDump class.

Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyDumpObjects</td>
<td>Represents the set of TMyDumpObject.</td>
</tr>
</tbody>
</table>

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyDumpObject</td>
<td>Specifies the object enumeration.</td>
</tr>
</tbody>
</table>

5.20.1 Classes

Classes in the MySQL unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyDump</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>TMyDumpOptions</td>
<td>This class allows setting up the behaviour of the TMyDump class.</td>
</tr>
</tbody>
</table>
For a list of all members of this type, see TMyDump members.

Unit
MyDump

Syntax

TMyDump = class(TDADump);

Remarks
Serves to store a database or its parts as a script and also to restore database from received script.

TMyDump behaviour is similar to mysqldump program.

Use TMyDump.Objects and TDADump.TableNames properties to specify the list of objects to be stored. By default, only tables structure and data are stored.

To generate a script call TDADump.Backup or TDADump.BackupQuery method. Resulted script can be viewed in TDADump.SQL.

TMyDump works on the client side. It causes large network loading. To backup database on the server side use TMyBackup. Unlike TMyBackup, TMyDump component allows to store not only tables, but database structure including users' rights.

Inheritance Hierarchy
TDADump
  TMyDump

See Also
• TDADump.Backup
• TDADump.Restore
• TMyBackup

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.20.1.1.1 Members

TMyDump 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 (inherited from <strong>TDADump</strong>)</td>
<td>Used to display the statement that is being executed and the values and types of its parameters.</td>
</tr>
<tr>
<td>Objects</td>
<td>Used to set the enumeration of object types that will be described on calling <strong>TDADump.Backup</strong>.</td>
</tr>
<tr>
<td>Options</td>
<td>Specifies the behaviour of the TMyDump component.</td>
</tr>
<tr>
<td>SQL (inherited from <strong>TDADump</strong>)</td>
<td>Used to set or get the dump script.</td>
</tr>
<tr>
<td>StoredProcNames</td>
<td>Holds the list of procedures which will be used in the script.</td>
</tr>
<tr>
<td>TableNames (inherited from <strong>TDADump</strong>)</td>
<td>Used to set the names of the tables to dump.</td>
</tr>
<tr>
<td>TriggerNames</td>
<td>Holds the list of triggers which will be used in the script.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Backup</strong> (inherited from <strong>TDADump</strong>)</td>
<td>Dumps database objects to the <strong>TDADump.SQL</strong> property.</td>
</tr>
<tr>
<td><strong>BackupQuery</strong> (inherited from <strong>TDADump</strong>)</td>
<td>Dumps the results of a particular query.</td>
</tr>
<tr>
<td><strong>BackupToFile</strong> (inherited from <strong>TDADump</strong>)</td>
<td>Dumps database objects to the specified file.</td>
</tr>
<tr>
<td><strong>BackupToStream</strong> (inherited from <strong>TDADump</strong>)</td>
<td>Dumps database objects to the stream.</td>
</tr>
<tr>
<td><strong>Restore</strong> (inherited from <strong>TDADump</strong>)</td>
<td>Executes a script contained in the SQL property.</td>
</tr>
</tbody>
</table>
### Properties

Properties of the **TMyDump** class.

For a complete list of the **TMyDump** class members, see the [TMyDump Members](#) topic.

---

### Published

<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>
</tbody>
</table>
### Debug (inherited from TDADump)

Used to display the statement that is being executed and the values and types of its parameters.

### Objects

Used to set the enumeration of object types that will be described on calling TDADump.Backup.

### Options

Specifies the behaviour of the TMyDump component.

### SQL (inherited from TDADump)

Used to set or get the dump script.

### StoredProcNames

Holds the list of procedures which will be used in the script.

### TableNames (inherited from TDADump)

Used to set the names of the tables to dump.

### TriggerNames

Holds the list of triggers which will be used in the script.

See Also
- TMyDump Class
- TMyDump Class Members

#### 5.20.1.1.2.1  Connection Property

Used to specify a connection object that will be used to connect to a data store.

**Class**

TMyDump

**Syntax**

```property
connection: TCustomMyConnection;
```

**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 TCustomMyConnection objects.

At runtime, set Connection property to reference an existing TCustomMyConnection object.

See Also
- TCustomMyConnection

Class
TMyDump

Syntax

```delphi
property Objects: TMyDumpObjects default [doTables, doViews, doData];
```

Remarks

Use the Object property to set the enumeration of object types that will be described on calling TDADump.Backup.

See Also
- TDADump.Backup
- TDADump.BackupToFile
- TDADump.BackupToStream
- TDADump.TableNames
- StoredProcNames
5.20.1.1.2.3 Options Property

Specifies the behaviour of the TMyDump component.

Class

TMyDump

Syntax

property Options: TMyDumpOptions;

Remarks

Set the properties of Options to specify the behaviour of a TMyDump 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>AddLock</td>
<td>Used to execute LOCK TABLE before data insertion.</td>
</tr>
<tr>
<td>CommitBatchSize</td>
<td>Used to add COMMIT statement to script after inserting every CommitBatchSize strings when dumping table data.</td>
</tr>
<tr>
<td>DisableKeys</td>
<td>Allows disabling keys check when inserting records.</td>
</tr>
<tr>
<td>HexBlob</td>
<td>Used to present BLOB values in hexadecimal notation.</td>
</tr>
<tr>
<td>InsertType</td>
<td>Used to specify the method of rows insertion into a table.</td>
</tr>
<tr>
<td>UseDelayedIns</td>
<td>Set to use INSERT DELAYED.</td>
</tr>
<tr>
<td>UseExtSyntax</td>
<td>Set to use extended syntax of INSERT on data insertion.</td>
</tr>
</tbody>
</table>

See Also

- Objects

© 1997-2021 Devart. All Rights Reserved.
5.20.1.1.2.4 StoredProcNames Property

Holds the list of procedures which will be used in the script.

Class

TMyDump

Syntax

```delphi
property StoredProcNames: string;
```

Remarks

Use the StoredProcNames property to hold the list of procedures which will be used in the script. Makes sense only on execution `TDADump.Backup`, `M:Devart.Dac.TDADump.BackupToFile(System.String)`, `M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream)` and `doStoredProcs set at the Objects`. Names are separated by comma or semicolon. If it has an empty value, all procs presented in the database will be processed.

See Also

- `TDADump.Backup`
- `TDADump.BackupToFile`
- `TDADump.BackupToStream`
- `Objects`
- `TCustomDAConnection.GetStoredProcNames`

5.20.1.1.2.5 TriggerNames Property

Holds the list of triggers which will be used in the script.

Class

TMyDump

Syntax

```delphi
property TriggerNames: string;
```

Remarks

© 1997-2021 Devart. All Rights Reserved.
Use the TriggerNames property to hold the list of triggers which will be used in the script. Makes sense only on execution of TMyDump.Backup, TMyDump.BackupToFile, TMyDump.BackupToStream, and doTriggers set at the Objects. Names are separated by comma or semicolon. If it has an empty value and the TMyDump/TableNames property is empty, then all triggers presented in the database will be processed. If it has an empty value and the TMyDump/TableNames property is not empty, then all triggers for the specified tables will be processed.

5.20.1.2 TMyDumpOptions Class

This class allows setting up the behaviour of the TMyDump class.

For a list of all members of this type, see TMyDumpOptions members.

Unit

MyDump

Syntax

TMyDumpOptions = class(TDADumpOptions);

Inheritance Hierarchy

TDADumpOptions
TMyDumpOptions

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><strong>AddLock</strong></td>
<td>Used to execute LOCK TABLE before data insertion.</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><strong>CommitBatchSize</strong></td>
<td>Used to add COMMIT statement to script after inserting every CommitBatchSize strings when dumping table data.</td>
</tr>
<tr>
<td><strong>CompleteInsert</strong> (inherited from <strong>TDADumpOptions</strong>)</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><strong>DisableKeys</strong></td>
<td>Allows disabling keys check when inserting records.</td>
</tr>
<tr>
<td><strong>GenerateHeader</strong> (inherited from <strong>TDADumpOptions</strong>)</td>
<td>Used to add a comment header to a script.</td>
</tr>
<tr>
<td><strong>HexBlob</strong></td>
<td>Used to present BLOB values in hexadecimal notation.</td>
</tr>
<tr>
<td><strong>InsertType</strong></td>
<td>Used to specify the method of rows insertion into a table.</td>
</tr>
<tr>
<td><strong>QuoteNames</strong> (inherited from <strong>TDADumpOptions</strong>)</td>
<td>Used for TDADump to quote all database object names in generated SQL statements.</td>
</tr>
<tr>
<td><strong>UseDelayedIns</strong></td>
<td>Set to use INSERT DELAYED.</td>
</tr>
<tr>
<td><strong>UseExtSyntax</strong></td>
<td>Set to use extended syntax of INSERT on data insertion.</td>
</tr>
</tbody>
</table>

Properties of the **TMyDumpOptions** class.

For a complete list of the **TMyDumpOptions** class members, see the **TMyDumpOptions Members** topic.

Published

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddDrop</strong>  (inherited from <code>TDADumpOptions</code>)</td>
<td>Used to add drop statements to a script before creating statements.</td>
</tr>
<tr>
<td><strong>AddLock</strong></td>
<td>Used to execute LOCK TABLE before data insertion.</td>
</tr>
<tr>
<td><strong>CommitBatchSize</strong></td>
<td>Used to add COMMIT statement to script after inserting every CommitBatchSize strings when dumping table data.</td>
</tr>
<tr>
<td><strong>CompleteInsert</strong>  (inherited from <code>TDADumpOptions</code>)</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><strong>DisableKeys</strong></td>
<td>Allows disabling keys check when inserting records.</td>
</tr>
<tr>
<td><strong>GenerateHeader</strong>  (inherited from <code>TDADumpOptions</code>)</td>
<td>Used to add a comment header to a script.</td>
</tr>
<tr>
<td><strong>HexBlob</strong></td>
<td>Used to present BLOB values in hexadecimal notation.</td>
</tr>
<tr>
<td><strong>InsertType</strong></td>
<td>Used to specify the method of rows insertion into a table.</td>
</tr>
<tr>
<td><strong>QuoteNames</strong>  (inherited from <code>TDADumpOptions</code>)</td>
<td>Used for TDADump to quote all database object names in generated SQL statements.</td>
</tr>
<tr>
<td><strong>UseDelayedIns</strong></td>
<td>Set to use INSERT DELAYED.</td>
</tr>
<tr>
<td><strong>UseExtSyntax</strong></td>
<td>Set to use extended syntax of INSERT on data insertion.</td>
</tr>
</tbody>
</table>

See Also

- `TMyDumpOptions Class`
- `TMyDumpOptions Class Members`

© 1997-2021 Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback
5.20.1.2.2.1 AddLock Property

Used to execute LOCK TABLE before data insertion.

Class

TMyDumpOptions

Syntax

```property`` AddLock: boolean `default` True;
```

Remarks

Use the AddLock property to execute LOCK TABLE before data insertion. Used only with `doData` in TMyDump.Objects.

© 1997-2021 Devart. All Rights Reserved.

5.20.1.2.2.2 CommitBatchSize Property

Used to add COMMIT statement to script after inserting every CommitBatchSize strings when dumping table data.

Class

TMyDumpOptions

Syntax

```property`` CommitBatchSize: integer `default` 0;
```

Remarks

Use the CommitBatchSize property to add COMMIT statement to script after inserting every CommitBatchSize strings when dumping table data. Use this property to boost the productivity when recovering large amounts of data.

© 1997-2021 Devart. All Rights Reserved.

Request Support DAC Forum Provide Feedback
5.20.1.2.2.3 DisableKeys Property

Allows disabling keys check when inserting records.

Class
TMyDumpOptions

Syntax

```property DisableKeys: boolean default False;```

Remarks
Add /*!40000 ALTER TABLE ... DISABLE KEYS */ before inserting data. Used only with doData in TMyDump.Objects.

© 1997-2021 Devart. All Rights Reserved.

5.20.1.2.2.4 HexBlob Property

Used to present BLOB values in hexadecimal notation.

Class
TMyDumpOptions

Syntax

```property HexBlob: boolean default False;```

Remarks
If the HexBlob property is True, the BLOB values are presented in hexadecimal notation.

© 1997-2021 Devart. All Rights Reserved.

5.20.1.2.2.5 InsertType Property

Used to specify the method of rows insertion into a table.

Class
TMyDumpOptions
## UseDelayedIns Property

Set to use INSERT DELAYED.

### Class

**TMyDumpOptions**

### Syntax

```delphi
property UseDelayedIns: boolean default False;
```

### Remarks

Set the UseDelayedIns property to use INSERT DELAYED. Used only with doData in **TMyDump.Objects**.

© 1997-2021 Devart. All Rights Reserved.

© 2021 Devart
**TMyDumpOptions**

**Syntax**

```plaintext
property UseExtSyntax: boolean default True;
```

**Remarks**

Set the UseExtSyntax property to use extended syntax of INSERT on data insertion. Used only with `doData` in `TMyDump.Objects`.

© 1997-2021 Devart. All Rights Reserved.

## 5.20.2 Types

Types in the **MyDump** unit.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyDumpObjects</td>
<td>Represents the set of <code>TMyDumpObject</code>.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

### 5.20.2.1 TMyDumpObjects Set

Represents the set of `TMyDumpObject`.

**Unit**

**MyDump**

**Syntax**

```plaintext
TMyDumpObjects = set of TMyDumpObject;
```

© 1997-2021 Devart. All Rights Reserved.
5.20.3 Enumerations

Enumerations in the MyDump unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyDumpObject</td>
<td>Specifies the object enumeration.</td>
</tr>
</tbody>
</table>

5.20.3.1 TMyDumpObject Enumeration

Specifies the object enumeration.

Unit

MyDump

Syntax

TMyDumpObject = (doDatabase, doUsers, doStoredProcs, doTables, doData, doViews, doTriggers);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>doData</td>
<td>Stores data from the tables. The list of the tables is specified in TDADump.TableName. If TableNames is not set, all tables are used.</td>
</tr>
<tr>
<td>doDatabase</td>
<td>Adds to the received script &quot;CREATE DATABASE...&quot; command.</td>
</tr>
<tr>
<td>doStoredProcs</td>
<td>Stores stored procedures structure in the script. The list of procedures is specified in TMyDump.StoredProcNames. If StoredProcNames is not set, all procedures are used.</td>
</tr>
<tr>
<td>doTables</td>
<td>Stores tables structure in the script. The list of tables is specified in TDADump.TableName. If TableNames is not set, all tables are used.</td>
</tr>
<tr>
<td>doTriggers</td>
<td>Stores queries for creating triggers in the script. The list of triggers is specified in TMyDump.TriggerNames. If TriggerNames is not set and the TMyDump.TableName property is empty, then all triggers of the database will be processed. If TriggerNames is not set and the TMyDump.TableName property...</td>
</tr>
</tbody>
</table>
is not empty, then all triggers for the specified tables will be processed.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>doUsers</strong></td>
<td>Stores user privileges in the script.</td>
</tr>
<tr>
<td><strong>doViews</strong></td>
<td>Stores queries for creating Views in a script. The Views list is taken from TableNames.</td>
</tr>
</tbody>
</table>

## 5.21 MyEmbConnection

This unit contains implementation of the TMyEmbConnection component.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyEmbConnection</td>
<td>A component for establishing connection to Embedded MySQL server.</td>
</tr>
</tbody>
</table>

### 5.21.1 Classes

Classes in the *MyEmbConnection* unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyEmbConnection</td>
<td>A component for establishing connection to Embedded MySQL server.</td>
</tr>
</tbody>
</table>

### 5.21.1.1 TMyEmbConnection Class

A component for establishing connection to Embedded MySQL server.

For a list of all members of this type, see [TMyEmbConnection](#) members.
Unit

`MyEmbConnection`

Syntax

```plaintext
TMyEmbConnection = class(TCustomMyConnection);
```

Remarks

TMyEmbConnection component is used to establish a connection to MySQL Embedded server and provide its enhanced support.

Using TMyEmbConnection allows refusing from using configuration file (my.ini) and also lets you handle MySQL server messages written into log files.

Inheritance Hierarchy

```
TCustomDAConnection
  TCustomMyConnection
    TMyEmbConnection
```

See Also

- `TCustomMyConnection`
- `TMyConnection`
- `Embedded Server`

© 1997-2021 Devart. All Rights Reserved.

5.21.1.1.1 Members

```
TMyEmbConnection class overview.
```

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BaseDir</code></td>
<td>Used to set the base path for MySQL Embedded server.</td>
</tr>
<tr>
<td><code>ClientVersion</code></td>
<td>(inherited from <code>TCustomMyConnection</code>) Contains the version of the MySQL Client library.</td>
</tr>
<tr>
<td><code>ConnectDialog</code></td>
<td>(inherited from <code>TCustomDAConnection</code>) Allows to link a</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>ConnectionTimeout</strong></td>
<td>Used to specify the amount of time to attempt to establish a connection. (inherited from <strong>TCustomMyConnection</strong>)</td>
</tr>
<tr>
<td><strong>ConnectString</strong></td>
<td>Used to specify the connection information, such as: UserName, Password, Server, etc. (inherited from <strong>TCustomDAConnection</strong>)</td>
</tr>
<tr>
<td><strong>ConvertEOL</strong></td>
<td>Allows customizing line breaks in string fields and parameters. (inherited from <strong>TCustomDAConnection</strong>)</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Used to specify a database name that is a default source of data for SQL queries once a connection is established. (inherited from <strong>TCustomMyConnection</strong>)</td>
</tr>
<tr>
<td><strong>DataDir</strong></td>
<td>Used to set the path to the data directory.</td>
</tr>
<tr>
<td><strong>InTransaction</strong></td>
<td>Indicates whether the transaction is active. (inherited from <strong>TCustomDAConnection</strong>)</td>
</tr>
<tr>
<td><strong>IsolationLevel</strong></td>
<td>Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection. (inherited from <strong>TCustomMyConnection</strong>)</td>
</tr>
<tr>
<td><strong>LoginPrompt</strong></td>
<td>Specifies whether a login dialog appears immediately before opening a new connection. (inherited from <strong>TCustomDAConnection</strong>)</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Specifies the behaviour of the TMyConnectionOptions object. (inherited from <strong>TCustomMyConnection</strong>)</td>
</tr>
<tr>
<td><strong>Params</strong></td>
<td>Used to specify the list of command line parameters for Embedded server.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Serves to supply a password for login. (inherited from <strong>TCustomDAConnection</strong>)</td>
</tr>
<tr>
<td><strong>Pooling</strong></td>
<td>Enables or disables using connection pool. (inherited from <strong>TCustomDAConnection</strong>)</td>
</tr>
<tr>
<td><strong>PoolingOptions</strong></td>
<td>Specifies the behaviour of connection pool. (inherited from <strong>TCustomDAConnection</strong>)</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>Serves to supply the server connection information. (inherited from <strong>TCustomDAConnection</strong>)</td>
</tr>
</tbody>
</table>
**ServerVersion** (inherited from **TCustomMyConnection**)  
Holds the version of MySQL server.

**ThreadId** (inherited from **TCustomMyConnection**)  
Used to return the thread ID of the current connection.

**Username** (inherited from **TCustomDAConnection**)  
Used to supply a user name for login.

### Methods

<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>AssignConnect</strong></td>
<td>(inherited from <strong>TCustomMyConnection</strong>) Shares database connection between the TCustomMyConnection components.</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>(inherited from <strong>TCustomMyConnection</strong>) Returns a new instance of TCustomMyDataSet class and associates it with this connection object.</td>
</tr>
<tr>
<td><strong>CreateSQL</strong></td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Creates a component for queries execution.</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 parameters.</td>
</tr>
<tr>
<td><strong>ExecProcEx</strong></td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Allows to execute a stored procedure or function.</td>
</tr>
<tr>
<td><strong>ExecSQL</strong></td>
<td>(inherited from <strong>TCustomMyConnection</strong>) Executes any SQL statement outside TMyQuery or TMyCommand components.</td>
</tr>
<tr>
<td><strong>ExecSQLEx</strong></td>
<td>(inherited from <strong>TCustomDAConnection</strong>) Executes any SQL statement outside the TQuery or TSQL components.</td>
</tr>
<tr>
<td><strong>GetCharsetNames</strong></td>
<td>(inherited from) Populates a string list with the names of available <strong>TCustomMyConnection</strong>.</td>
</tr>
<tr>
<td><strong>TCustomMyConnection</strong></td>
<td>charsets.</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>GetDatabaseNames</strong> (inherited from TCustomDACConnection)</td>
<td>Returns a database list from the server.</td>
</tr>
<tr>
<td><strong>GetExecuteInfo</strong> (inherited from TCustomMyConnection)</td>
<td>Returns the result of the last query execution.</td>
</tr>
<tr>
<td><strong>GetKeyFieldNames</strong> (inherited from TCustomDACConnection)</td>
<td>Provides a list of available key field names.</td>
</tr>
<tr>
<td><strong>GetStoredProcNames</strong> (inherited from TCustomDACConnection)</td>
<td>Returns a list of stored procedures from the server.</td>
</tr>
<tr>
<td><strong>GetTableNames</strong> (inherited from TCustomDACConnection)</td>
<td>Provides a list of available tables names.</td>
</tr>
<tr>
<td><strong>GetTriggerNames</strong> (inherited from TCustomMyConnection)</td>
<td>Returns a list of triggers from the server.</td>
</tr>
<tr>
<td><strong>MonitorMessage</strong> (inherited from TCustomDACConnection)</td>
<td>Sends a specified message through the TCustomDASQLMonitor component.</td>
</tr>
<tr>
<td><strong>Ping</strong> (inherited from TCustomDACConnection)</td>
<td>Used to check state of connection to the server.</td>
</tr>
<tr>
<td><strong>ReleaseSavepoint</strong> (inherited from TCustomMyConnection)</td>
<td>Releases the specified savepoint without affecting any work that has been performed after its creation.</td>
</tr>
<tr>
<td><strong>RemoveFromPool</strong> (inherited from TCustomDACConnection)</td>
<td>Marks the connection that should not be returned to the pool after disconnect.</td>
</tr>
<tr>
<td><strong>Rollback</strong> (inherited from TCustomDACConnection)</td>
<td>Discards all current data changes and ends transaction.</td>
</tr>
<tr>
<td><strong>RollbackToSavepoint</strong> (inherited from TCustomMyConnection)</td>
<td>Cancels all updates for the current transaction.</td>
</tr>
<tr>
<td><strong>Savepoint</strong> (inherited from TCustomMyConnection)</td>
<td>Defines a point in the transaction to which you can roll back later.</td>
</tr>
<tr>
<td><strong>StartTransaction</strong> (inherited from TCustomDACConnection)</td>
<td>Begins a new user transaction.</td>
</tr>
</tbody>
</table>

**Events**
Properties of the TMyEmbConnection class.

For a complete list of the TMyEmbConnection class members, see the TMyEmbConnection Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientVersion</td>
<td>Contains the version of the MySQL Client library.</td>
</tr>
<tr>
<td>ConnectDialog</td>
<td>Allows to link a TCustomConnectDialog component.</td>
</tr>
<tr>
<td>ConnectionTimeout</td>
<td>Used to specify the amount of time to attempt to establish a connection.</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>Database</td>
<td>Used to specify a database name that is a default source of data for SQL queries</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InTransaction</td>
<td>Indicates whether the transaction is active.</td>
</tr>
<tr>
<td>IsolationLevel</td>
<td>Used to specify extent to which all outside transactions interfere with subsequent transactions of current connection.</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 behaviour of the TMyConnectionOptions object.</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>ServerVersion</td>
<td>Holds the version of MySQL server.</td>
</tr>
<tr>
<td>ThreadId</td>
<td>Used to return the thread ID of the current connection.</td>
</tr>
<tr>
<td>Username</td>
<td>Used to supply a user name for login.</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseDir</td>
<td>Used to set the base path for MySQL Embedded server.</td>
</tr>
<tr>
<td>DataDir</td>
<td>Used to set the path to the data directory.</td>
</tr>
<tr>
<td>Params</td>
<td>Used to specify the list of command line parameters for Embedded server.</td>
</tr>
</tbody>
</table>
5.21.1.1.2.1  BaseDir Property

Used to set the base path for MySQL Embedded server.

Class
TMyEmbConnection

Syntax

| property BaseDir: string stored False; |

Remarks

Use the BaseDir property to set the base path for MySQL Embedded server. All paths are usually resolved relative to this. Corresponds to --basedir parameter. See MySQL Reference Manual for detailed description.

The default value is '.' and all auxiliary files must be located in the same folder with the executable file of the project.

See Also
- DataDir
- Params
- Embedded Server
Syntax

```
property DataDir: string stored False;
```

Remarks

Use the DataDir property to set the path to the data directory. Corresponds to --datadir parameter. See MySQL Reference Manual for a detailed description.

The default value is 'data' and all data files must be in the subfolder data by the path specified in BaseDir.

It is convenient to use this property for cases when the program is started from ReadOnly data carrier (CD-ROM, network etc). Also its usage can be suitable to separate data of different users.

See Also

- `BaseDir`
- `Params`
- `Embedded Server`

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback

5.21.1.2.3  Params Property

Used to specify the list of command line parameters for Embedded server.

Class

`TMyEmbConnection`

Syntax

```
property Params: TStrings;
```

Remarks

Use the Params property to specify the list of command line parameters for Embedded server.

Pay attention that all paths must be set through "/" but not "/".

Params property also stores values for BaseDir and DataDir.
If no parameters are set in Params property its value will be obtained from the file of configuration of EmbServer (my.ini or my.cnf). Please see MySQL Reference for details.

libmysqld library is reloaded only on changing parameters.

Note, parameters names are case-sensitive.

See Also
- BaseDir
- DataDir
- Embedded Server

© 1997-2021 Devart. All Rights Reserved.

5.21.1.1.3 Events
Events of the TMyEmbConnection class.
For a complete list of the TMyEmbConnection class members, see the TMyEmbConnection Members topic.

Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnectionLost (inherited from TCustomDACConnection)</td>
<td>This event occurs when connection was lost.</td>
</tr>
<tr>
<td>OnError (inherited from TCustomDACConnection)</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>OnLog</td>
<td>Occurs when MySQL server writes down to General Query Log.</td>
</tr>
<tr>
<td>OnLogError</td>
<td>Occurs when MySQL server writes down to Error Log.</td>
</tr>
</tbody>
</table>

See Also
OnLog Event

Occurs when MySQL server writes down to General Query Log.

Class

TMyEmbConnection

Syntax

```property``

```OnLog: TMyLogEvent;```

Remarks

The OnLog event occurs when MySQL server writes down to General Query Log, the same as on using `--log` option. See MySQL Reference Manual for detailed description.

On assigning handler for OnLog event MySQL server does not make output to common log-file.

This event is available only for Win32.

See Also

- OnLogError
- Embedded Server
### Syntax

```plaintext
property OnLogError: TMyLogEvent;
```

### Remarks

The `OnLogError` event occurs when MySQL server writes down to Error Log, the same as on using `--log-error` option. See MySQL Reference Manual for detailed description. It is convenient to use `OnLogError` event to search errors in the server configuration, as often after error message is displayed an application is terminated.

On assigning handler for `OnLogError` event MySQL server does not make output to common error log-file.

This event is available only for Win32.

### See Also
- `OnLog`
- `Embedded Server`

---

### 5.22 MyLoader

This unit contains implementation of the TMyLoader component.

### Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyColumn</td>
<td>A component representing the attributes for column loading.</td>
</tr>
<tr>
<td>TMyLoader</td>
<td>TMyLoader allows to load external data into the server database.</td>
</tr>
</tbody>
</table>

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyLoaderOptions</td>
<td>Represents the set of <code>TMyLoaderOption</code>.</td>
</tr>
</tbody>
</table>
Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyDuplicateKeys</td>
<td>Specifies the way conflicts with duplicated key values will be resolved.</td>
</tr>
<tr>
<td>TMyLoaderOption</td>
<td>Specifies the behaviour of a TMyLoader object.</td>
</tr>
</tbody>
</table>

5.22.1 Classes

Classes in the **MyLoader** unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyColumn</td>
<td>A component representing the attributes for column loading.</td>
</tr>
<tr>
<td>TMyLoader</td>
<td>TMyLoader allows to load external data into the server database.</td>
</tr>
</tbody>
</table>

5.22.1.1 TMyColumn Class

A component representing the attributes for column loading.

For a list of all members of this type, see [TMyColumn members](#).

Unit

**MyLoader**

Syntax

```
TMyColumn = class(TDAColumn);
```

Remarks
Each TMyLoader uses TDAColumns to maintain a collection of TMSColumn objects. TMyColumn object represents the attributes for column loading. Every TMyColumn object corresponds to one of the table fields with the same name as its TDAColumn.Name property.

To create columns at design-time use column editor of the TMyLoader component.

Inheritance Hierarchy

TDAColumn
  TMyColumn

See Also
- TMyLoader
- TDAColumns

<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>
</tbody>
</table>

TMyLoader Class

TMyLoader allows to load external data into the server database.

For a list of all members of this type, see TMyLoader members.
```cpp
TMyLoader = class(TDALoader);
```

## Remarks

TMyLoader serves for fast loading of data to the server.

TMyLoader work is based on generation INSERT statements which insert data by several rows for one time (see `TMyLoader.RowsPerQuery`).

This class is used to simplify INSERT statements generation.

To affect on performance set MyLoader.Connection.Options.Compression property to True.

The quicker way is to generate text file on the client side and load it using `TMyBackup.Restore` method with Mode = bmText.

### Inheritance Hierarchy

- `TDALoader`
- `TMyLoader`

### See Also

- `TMyBackup`

---

5.22.1.2.1 Members

**TMyLoader** class overview.

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Columns</strong> (inherited from <code>TDALoader</code>)</td>
<td>Used to add a <code>TDAColumn</code> object for each field that will be loaded.</td>
</tr>
<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>DuplicateKeys</strong></td>
<td>Used to specify in what way conflicts with duplicated key values will be resolved.</td>
</tr>
</tbody>
</table>
Properties of the **TMyLoader** class.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options</strong></td>
<td>Specifies the behaviour of TMyLoader object.</td>
</tr>
<tr>
<td><strong>RowsPerQuery</strong></td>
<td>Used to get or set the number of rows that will be send to the server for one time.</td>
</tr>
<tr>
<td><strong>TableName</strong> (inherited from <strong>TDALoader</strong>)</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><strong>CreateColumns</strong> (inherited from <strong>TDALoader</strong>)</td>
<td>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> (inherited from <strong>TDALoader</strong>)</td>
<td>Starts loading data.</td>
</tr>
<tr>
<td><strong>LoadFromDataSet</strong> (inherited from <strong>TDALoader</strong>)</td>
<td>Loads data from the specified dataset.</td>
</tr>
<tr>
<td><strong>PutColumnData</strong> (inherited from <strong>TDALoader</strong>)</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><strong>OnGetColumnData</strong> (inherited from <strong>TDALoader</strong>)</td>
<td>Occurs when it is needed to put column values.</td>
</tr>
<tr>
<td><strong>OnProgress</strong> (inherited from <strong>TDALoader</strong>)</td>
<td>Occurs if handling data loading progress of the <strong>TDALoader.LoadFromDataSet</strong> method is needed.</td>
</tr>
<tr>
<td><strong>OnPutData</strong> (inherited from <strong>TDALoader</strong>)</td>
<td>Occurs when putting loading data by rows is needed.</td>
</tr>
</tbody>
</table>
For a complete list of the **TMyLoader** class members, see the [TMyLoader Members](#) topic.

### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Columns</strong></td>
<td>(inherited from <strong>TDALoader</strong>) Used to add a <strong>TDAColumn</strong> object for each field that will be loaded.</td>
</tr>
<tr>
<td><strong>TableName</strong></td>
<td>(inherited from <strong>TDALoader</strong>) Used to specify the name of the table to which data will be loaded.</td>
</tr>
</tbody>
</table>

### Published

<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>DuplicateKeys</strong></td>
<td>Used to specify in what way conflicts with duplicated key values will be resolved.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Specifies the behaviour of <strong>TMyLoader</strong> object.</td>
</tr>
<tr>
<td><strong>RowsPerQuery</strong></td>
<td>Used to get or set the number of rows that will be send to the server for one time.</td>
</tr>
</tbody>
</table>

**See Also**
- [TMyLoader Class](#)
- [TMyLoader Class Members](#)

© 1997-2021 Devart. All Rights Reserved.  

5.22.1.2.2.1 Connection Property

Used to specify a connection object that will be used to connect to a data store.

**Class**

[TMyLoader](#)
## Syntax

**property** Connection: **TCustomMyConnection**;

## 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 TMyConnection objects.

At runtime, set the Connection property to reference an existing TMyConnection object.

### See Also
- **TCustomMyConnection**

## Class

**TMyLoader**

## Syntax

**property** DuplicateKeys: **TMyDuplicateKeys** default dkNone;

## Remarks

Use the DuplicateKeys property to specify in what way conflicts with duplicated key values will be resolved.

## Class

**TMyLoader**

## Syntax

**property** DuplicateKeys: **TMyDuplicateKeys** default dkNone;

## Remarks

Use the DuplicateKeys property to specify in what way conflicts with duplicated key values will be resolved.

### See Also
- **TMyLoader**

## Class

**TMyLoader**
**TMyLoader**

**Syntax**

```plaintext
property Options: TMyLoaderOptions default [];
```

**Remarks**

Set the properties of Options to specify the behaviour of a TMyLoader object.

---

**5.22.1.2.4 RowsPerQuery Property**

Used to get or set the number of rows that will be send to the server for one time.

**Class**

**TMyLoader**

**Syntax**

```plaintext
property RowsPerQuery: integer default 0;
```

**Remarks**

Use the RowsPerQuery property to get or set the number of rows that will be send to the server for one time. The default value is 0. In this case rows will be grouped by 16Kb (the default value of net_buffer_length).

---

**5.22.2 Types**

Types in the **MyLoader** unit.

**Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyLoaderOptions</td>
<td>Represents the set of TMyLoaderOption.</td>
</tr>
</tbody>
</table>
5.22.2.1 TMyLoaderOptions Set

Represents the set of TMyLoaderOption.

Unit
MyLoader

Syntax

\[
\text{TMyLoaderOptions} = \text{set of TMyLoaderOption};
\]

5.22.3 Enumerations

Enumerations in the MyLoader unit.

Enumerations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyDuplicateKeys</td>
<td>Specifies the way conflicts with duplicated key values will be resolved.</td>
</tr>
<tr>
<td>TMyLoaderOption</td>
<td>Specifies the behaviour of a TMyLoader object.</td>
</tr>
</tbody>
</table>

5.22.3.1 TMyDuplicateKeys Enumeration

Specifies the way conflicts with duplicated key values will be resolved.

Unit
MyLoader

Syntax

\[
\text{TMyDuplicateKeys} = (\text{dkNone}, \text{dkIgnore}, \text{dkReplace});
\]
5.22.3.2 TMyLoaderOption Enumeration

Specifies the behaviour of a TMyLoader object.

Unit

MyLoader

Syntax

TMyLoaderOption = (loLock, loDelayed);

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>loDelayed</td>
<td>Uses INSERT DELAYED syntax.</td>
</tr>
<tr>
<td>loLock</td>
<td>Locks tables while inserting data.</td>
</tr>
</tbody>
</table>

5.23 MyScript

This unit contains implementation of the TMyScript component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyScript</td>
<td>A component for executing several SQL statements one</td>
</tr>
</tbody>
</table>
5.23.1 Classes

Classes in the MyScript unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyScript</td>
<td>A component for executing several SQL statements one by one.</td>
</tr>
</tbody>
</table>

5.23.1.1 TMyScript Class

A component for executing several SQL statements one by one.

For a list of all members of this type, see TMyScript members.

Unit

MyScript

Syntax

\[ \text{TMyScript} = \text{class}(\text{TDAScript}); \]

Remarks

Often it is necessary to execute several SQL statements one by one. Known way is using a lot of components such as TMyCommand. Usually it is not a good solution. With only one TMyScript component you can execute several SQL statements as one. This sequence of statements is named script. To separate single statements use semicolon (;), slash (/), and DELIMITER . Note that slash must be the first character in line.

Errors that occur while execution can be processed in the TDAScript.OnError event handler. By default, on error TMyScript shows exception and continues execution.
If you need to create several Stored Procedures (Functions) at a single script, use slash (/) to separate commands for creating stored procedures.

Inheritance Hierarchy

TDAScript

TMyScript

See Also

- TMyCommand

© 1997-2021 Devart. All Rights Reserved.

TMyScript 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>DataSet</td>
<td>Used to retrieve the results of SELECT statements execution inside a script.</td>
</tr>
<tr>
<td>Debug</td>
<td>(inherited from TDAScript) Used to display the script execution and all its parameter values.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>(inherited from TDAScript) Used to set the delimiter string that separates script statements.</td>
</tr>
<tr>
<td>EndLine</td>
<td>(inherited from TDAScript) Used to get the current statement last line number in a script.</td>
</tr>
<tr>
<td>EndOffset</td>
<td>(inherited from TDAScript) Used to get the offset in the last line of the current statement.</td>
</tr>
<tr>
<td>EndPos</td>
<td>(inherited from TDAScript) Used to get the end position of the current statement.</td>
</tr>
<tr>
<td>Macros</td>
<td>(inherited from TDAScript) Used to change SQL script</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BreakExec</strong></td>
<td>(inherited from TDAScript) Stops script execution.</td>
</tr>
<tr>
<td><strong>ErrorOffset</strong></td>
<td>(inherited from TDAScript) Used to get the offset of the statement if the Execute method raised an exception.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>(inherited from TDAScript) Executes a script.</td>
</tr>
<tr>
<td><strong>ExecuteFile</strong></td>
<td>(inherited from TDAScript) Executes SQL statements contained in a file.</td>
</tr>
<tr>
<td><strong>ExecuteNext</strong></td>
<td>(inherited from TDAScript) Executes the next statement in the script and then stops.</td>
</tr>
<tr>
<td><strong>ExecuteStream</strong></td>
<td>(inherited from TDAScript) Executes SQL statements contained in a stream object.</td>
</tr>
<tr>
<td><strong>FindMacro</strong></td>
<td>(inherited from TDAScript) Finds a macro with the specified name.</td>
</tr>
<tr>
<td><strong>MacroByName</strong></td>
<td>(inherited from TDAScript) Finds a macro with the specified name.</td>
</tr>
</tbody>
</table>

### Events
### Properties of the TMyScript class.

For a complete list of the TMyScript class members, see the [TMyScript Members](#) topic.

#### Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 last line of the current statement.</td>
</tr>
<tr>
<td>EndPos (inherited from TDAScript)</td>
<td>Used to get the end position of the current statement.</td>
</tr>
<tr>
<td>StartLine (inherited from TDAScript)</td>
<td>Used to get the current statement start line number in a script.</td>
</tr>
<tr>
<td>StartOffset (inherited from TDAScript)</td>
<td>Used to get the offset in the first line of the current statement.</td>
</tr>
<tr>
<td>StartPos (inherited from TDAScript)</td>
<td>Used to get the start position of the current statement in a script.</td>
</tr>
<tr>
<td>Statements (inherited from TDAScript)</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>Connection</td>
<td>Used to specify a connection object that will be used to connect to a data store.</td>
</tr>
<tr>
<td>DataSet</td>
<td>Used to retrieve the results of SELECT statements execution inside a script.</td>
</tr>
<tr>
<td>Debug (inherited from <strong>TDAScript</strong>)</td>
<td>Used to display the script execution and all its parameter values.</td>
</tr>
<tr>
<td>Delimiter (inherited from <strong>TDAScript</strong>)</td>
<td>Used to set the delimiter string that separates script statements.</td>
</tr>
<tr>
<td>Macros (inherited from <strong>TDAScript</strong>)</td>
<td>Used to change SQL script text in design- or run-time easily.</td>
</tr>
<tr>
<td>SQL (inherited from <strong>TDAScript</strong>)</td>
<td>Used to get or set script text.</td>
</tr>
<tr>
<td>UseOptimization</td>
<td>Used to unit small queries into blocks for block executing if possible.</td>
</tr>
</tbody>
</table>

### See Also
- **TMyScript Class**
- **TMyScript Class Members**

© 1997-2021 Devart. All Rights Reserved.

© 2021 Devart

5.23.1.2.1 Connection Property

Used to specify a connection object that will be used to connect to a data store.

**Class**

**TMyScript**

**Syntax**

```property` Connection: TCustomMyConnection;```
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 TMyConnection objects.

At run-time, set Connection property to reference an existing TMyConnection object.

See Also

- TCustomMyConnection

5.23.1.2.2  DataSet Property

Used to retrieve the results of SELECT statements execution inside a script.

Class

TMyScript

Syntax

```property dataSet: TCustomMyDataSet;```

Remarks

Use the DataSet property if you need to retrieve the results of SELECT statements execution inside a script.

See Also

- TDAScript.Execute

5.23.1.2.3  UseOptimization Property

Used to unit small queries into blocks for block executing if possible.

Class

TMyScript
Syntax

```
property UseOptimization: boolean;
```

Remarks

Use the UseOptimization property to unit small queries into blocks for block executing if possible. The UseOptimization option does not affect the `TDAScript.ExecuteNext` method performance. It works only for the `TDAScript.Execute` method.

© 1997-2021 Devart. All Rights Reserved.

5.24 MyServerControl

This unit contains implementation of the TMyServerControl component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyServerControl</td>
<td>A component for server control and standard service tasks execution.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

5.24.1 Classes

Classes in the MyServerControl unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyServerControl</td>
<td>A component for server control and standard service tasks execution.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.
5.24.1.1 TMyServerControl Class

A component for server control and standard service tasks execution.

For a list of all members of this type, see TMyServerControl members.

Unit
TMyServerControl

Syntax
TMyServerControl = class(TCustomMyDataSet);

Remarks
TMyServerControl is a direct descendant of TCustomMyDataSet component. It serves to control the server and execute standard service tasks. List of the functions can be divided in the following groups:

- Create and delete a database (TMyServerControl.CreateDatabase, TMyServerControl.DropDatabase);
- Serves to manage MySQL services. available only for Windows (TMyServerControl.ServiceStart, TMyServerControl.ServiceStop, TMyServerControl.GetServiceNames, TMyServerControl.ServiceStatus);
- Manage MySQL server system variables (TMyServerControl.Variables, TMyServerControl.ShowVariables)
- Flush server data (TMyServerControl.Flush);
- Manage tables (TMyServerControl.AnalyzeTable, TMyServerControl.OptimizeTable, TMyServerControl.CheckTable, TMyServerControl.RepairTable);
- Manage processes (TMyServerControl.ShowProcessList, TMyServerControl.KillProcess);
- Obtain current information about the server (TMyServerControl.ShowStatus).

The last three groups show report as DataSet.

Inheritance Hierarchy

TMemDataSet
  ^
  TCustomDADataSet
    ^
    TCustomMyDataSet
      ^
      TMyServerControl

See Also
**TMyQuery**
© 1997-2021 Devart. All Rights Reserved.

5.24.1.1.1 Members

**TMyServerControl** class overview.

## Properties

<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>CommandTimeout</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Used to specify the amount of time to attempt execution of a command.</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>Used to specify a connection object to use for connecting to a data store.</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>Used to display executing statement.</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>Encryption</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>FetchAll</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Defines whether to request all records of the query from database server when the dataset is being opened.</td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td><strong>Inherited From</strong></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>InsertId</td>
<td>TCustomMyDataSet</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>LocalConstraints</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LocalUpdate</td>
<td>TMemDataSet</td>
</tr>
<tr>
<td>LockMode</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>MacroCount</td>
<td>TCustomDADataset</td>
</tr>
<tr>
<td>Property</td>
<td>Inherited From</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------</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>TCustomMyDataSet</td>
</tr>
<tr>
<td>ParamCheck</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>ParamCount</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Params</td>
<td>TCustomDADataSet</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>SmartFetch</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td><strong>Table 1: Data Access Components</strong></td>
<td></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>SQL</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>memory consumption.</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 the record count when opening a dataset.</td>
</tr>
<tr>
<td><strong>SQLRefresh</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>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><strong>SQLUpdate</strong> (inherited from <strong>TCustomDADataset</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>TableNames</strong></td>
<td>Holds a list of tables that will be processed by the <strong>TMyServerControl.AnalyzeTable</strong>, <strong>TMyServerControl.OptimizeTable</strong>, <strong>TMyServerControl.CheckTable</strong>, and <strong>TMyServerControl.RepairTable</strong> methods.</td>
</tr>
<tr>
<td><strong>UniDirectional</strong> (inherited from <strong>TCustomDADataset</strong>)</td>
<td>Used if an application does not need bidirectional access to records in the result set.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>UpdateObject</strong> (inherited from TCustomMyDataSet)</td>
<td>Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.</td>
</tr>
<tr>
<td><strong>UpdateRecordTypes</strong> (inherited from TMemDataSet)</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 TMemDataSet)</td>
<td>Used to check the status of the cached updates buffer.</td>
</tr>
<tr>
<td><strong>Variables</strong></td>
<td>Used to get or set the values of MySQL Server system variables.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AddWhere</strong> (inherited from TCustomDADataset)</td>
<td>Adds condition to the WHERE clause of SELECT statement in the SQL property.</td>
</tr>
<tr>
<td><strong>AnalyzeTable</strong></td>
<td>Analyzes and stores the key distribution for the table</td>
</tr>
<tr>
<td><strong>ApplyRange</strong> (inherited from TMemDataSet)</td>
<td>Applies a range to the dataset.</td>
</tr>
<tr>
<td><strong>ApplyUpdates</strong> (inherited from TMemDataSet)</td>
<td>Overloaded. Writes dataset's pending cached updates to a database.</td>
</tr>
<tr>
<td><strong>BreakExec</strong> (inherited from TCustomDADataset)</td>
<td>Breaks execution of the SQL statement on the server.</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>CheckTable</strong></td>
<td>Checks MyISAM and InnoDB tables for failures.</td>
</tr>
<tr>
<td><strong>CommitUpdates</strong> (inherited from TMemDataSet)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td><strong>CreateBlobStream</strong> (inherited from TCustomDADataset)</td>
<td>Used to obtain a stream for reading data from or writing</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CreateDatabase</td>
<td>Creates a new database with the specified name.</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>DropDatabase</td>
<td>Drops the specified database.</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</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FindParam</td>
<td>KeyValues parameter.</td>
</tr>
<tr>
<td></td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>Flush</td>
<td>Flushes server data from the memory to disk.</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>GetFieldEnum</td>
<td>Retrieve the list of acceptable values for a specified field given by the FieldName parameter.</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>GetServiceNames</td>
<td>Populates a list of MySQL services at the server to which Connection is connected.</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>KillProcess</td>
<td>Breaks the execution of the specified process.</td>
</tr>
<tr>
<td>Locate</td>
<td>Overloaded. Searches a dataset for a specific record and positions the cursor on it.</td>
</tr>
</tbody>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>Overloaded. Locks the current record for the current connection.</td>
</tr>
<tr>
<td>LockTable</td>
<td>Locks table for the current connection.</td>
</tr>
<tr>
<td>MacroByName</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>OpenNext</td>
<td>Opens next rowset in the statement.</td>
</tr>
<tr>
<td>OptimizeTable</td>
<td>Packs tables deleting unused places from the file at the disk and defragmenting it.</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>RefreshQuick</td>
<td>Retrieves changes posted to the server by another clients on the client side quickly.</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>Actualizes field values for the current record.</td>
</tr>
<tr>
<td>RepairTable</td>
<td>Repairs specified tables.</td>
</tr>
<tr>
<td>RestoreSQL</td>
<td>Restores the SQL property modified by AddWhere and SetOrderBy.</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>SaveSQL</td>
<td>Saves the SQL property value to BaseSQL.</td>
</tr>
<tr>
<td>SaveToXML</td>
<td>Overloaded. Saves the current dataset data to a file.</td>
</tr>
</tbody>
</table>

© 2021 Devart
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceStart</td>
<td>Starts the specified service at the server to which Connection is connected.</td>
</tr>
<tr>
<td>ServiceStatus</td>
<td>Returns a status of the specified service at the server to which Connection is connected.</td>
</tr>
<tr>
<td>ServiceStop</td>
<td>Stops the specified service at the server to which Connection is connected.</td>
</tr>
<tr>
<td>SetOrderBy</td>
<td>Builds an ORDER BY clause of a SELECT statement.</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>ShowProcessList</td>
<td>Shows the list of the processes.</td>
</tr>
<tr>
<td>ShowStatus</td>
<td>Shows the current state of the server.</td>
</tr>
<tr>
<td>ShowVariables</td>
<td>Shows the values of some MySQL system variables that are in effect for the current connection.</td>
</tr>
<tr>
<td>SQLSaved</td>
<td>Determines if the SQL property value was saved to the BaseSQL property.</td>
</tr>
<tr>
<td>UnLock</td>
<td>Releases a record lock.</td>
</tr>
<tr>
<td>UnLockTable</td>
<td>Releases a table locked by the TCustomMyDataSet.LockTable method.</td>
</tr>
</tbody>
</table>

or a stream in the XML format compatible with ADO format.
### 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> (inherited from <code>TCustomDADataset</code>)</td>
<td>Occurs after a component has executed a query to database.</td>
</tr>
<tr>
<td><strong>AfterFetch</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Occurs after dataset finishes fetching data from server.</td>
</tr>
<tr>
<td><strong>AfterUpdateExecute</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Occurs after executing insert, delete, update, lock and refresh operations.</td>
</tr>
<tr>
<td><strong>BeforeFetch</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Occurs before dataset is going to fetch block of records from the server.</td>
</tr>
<tr>
<td><strong>BeforeUpdateExecute</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Occurs before executing insert, delete, update, lock, and refresh operations.</td>
</tr>
<tr>
<td><strong>OnUpdateError</strong> (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><strong>OnUpdateRecord</strong> (inherited from <code>TMemDataSet</code>)</td>
<td>Occurs when a single update component can not handle the updates.</td>
</tr>
</tbody>
</table>
Properties of the `TMyServerControl` class.

For a complete list of the `TMyServerControl` class members, see the `TMyServerControl Members` topic.

## Public

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BaseSQL</strong> (inherited from <code>TCustomDADataset</code>)</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>CommandTimeout</strong> (inherited from <code>TCustomMyDataSet</code>)</td>
<td>Used to specify the amount of time to attempt execution of a command.</td>
</tr>
<tr>
<td><strong>Conditions</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Used to add WHERE conditions to a query</td>
</tr>
<tr>
<td><strong>DataTypeMap</strong> (inherited from <code>TCustomDADataset</code>)</td>
<td>Used to set data type mapping rules</td>
</tr>
<tr>
<td><strong>DetailFields</strong> (inherited from <code>TCustomDADataset</code>)</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 <code>TCustomDADataset</code>)</td>
<td>Used to keep dataset opened after connection is closed.</td>
</tr>
<tr>
<td><strong>Encryption</strong> (inherited from <code>TCustomMyDataSet</code>)</td>
<td>Used to specify encryption options in a dataset.</td>
</tr>
<tr>
<td><strong>FetchAll</strong> (inherited from <code>TCustomMyDataSet</code>)</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 <code>TCustomDADataset</code>)</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 <code>TCustomDADataset</code>)</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><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>InsertId</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Returns the ID generated for an AUTO_INCREMENT column by the previous query.</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 statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.</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>LockMode</strong></td>
<td>(inherited from <strong>TCustomMyDataSet</strong>) Specifies when to perform locking of an editing record.</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</td>
</tr>
<tr>
<td>Reference</td>
<td>MasterSource (inherited from <code>TCustomDADataSet</code>)</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Options (inherited from <code>TCustomMyDataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>Uses the behaviour of <code>TCustomMyDataSet</code> object.</td>
</tr>
<tr>
<td></td>
<td>ParamCheck (inherited from <code>TCustomDADataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>ParamCount (inherited from <code>TCustomDADataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>Params (inherited from <code>TCustomDADataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>Prepared (inherited from <code>TMemDataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>Ranged (inherited from <code>TMemDataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>ReadOnly (inherited from <code>TCustomDADataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>RefreshOptions (inherited from <code>TCustomDADataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>RowsAffected (inherited from <code>TCustomDADataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>SmartFetch (inherited from <code>TCustomMyDataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>SQL (inherited from <code>TCustomDADataSet</code>)</td>
</tr>
<tr>
<td></td>
<td>SQLDelete (inherited from <code>TCustomDADataSet</code>)</td>
</tr>
</tbody>
</table>
when applying a deletion to a record.

<p>| SQLInsert (inherited from TCustomDADataSet) | Used to specify the SQL statement that will be used when applying an insertion to a dataset. |
| SQLLock (inherited from TCustomDADataSet) | Used to specify a SQL statement that will be used to perform a record lock. |
| SQLRecCount (inherited from TCustomDADataSet) | Used to specify the SQL statement that is used to get the record count when opening a dataset. |
| 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 TCustomMyDataSet) | Used to specify an update object component which provides SQL statements that perform updates of the 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. |
| Variables | Used to get or set the values of MySQL Server system variables. |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Used to specify a connection object to use for connecting to a data store.</td>
</tr>
<tr>
<td>Debug</td>
<td>Used to display executing statement.</td>
</tr>
<tr>
<td>TableNames</td>
<td>Holds a list of tables that will be processed by the TMyServerControl.AnalyzeTable, TMyServerControl.OptimizeTable, TMyServerControl.CheckTable, TMyServerControl.RepairTable methods.</td>
</tr>
</tbody>
</table>

See Also
- TMyServerControl Class
- TMyServerControl Class Members

5.24.1.1.2.1 Connection Property

Used to specify a connection object to use for connecting to a data store.

Class
TMyServerControl

Syntax
```
property Connection: TCustomMyConnection;
```

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 TMyConnection objects.

At run-time, set Connection property to reference an existing TMyConnection object.

See Also
### Debug Property

Used to display executing statement.

**Class**

**TMyServerControl**

**Syntax**

```pascal
property Debug: boolean;
```

**Remarks**

Set the Debug property to True to display executing statement.

The default value is False.

**Note:** To enable debug form display you should explicitly include MyDacVcl unit into your project.

**See Also**

- **TCustomDADataset.Debug**

### TableNames Property

Holds a list of tables that will be processed by the `AnalyzeTable`, `OptimizeTable`, `CheckTable`, `RepairTable` methods.

**Class**

**TMyServerControl**

**Syntax**

```pascal
property TableNames: string;
```

**Remarks**
Contains a list of tables that will be processed by the AnalyzeTable, OptimizeTable, CheckTable, RepairTable methods.

Table names are separated by comma or semicolon. If it has an empty value, all tables presented in the database will be processed.

See Also
- AnalyzeTable
- OptimizeTable
- CheckTable
- RepairTable
- TCustomDACConnection.GetTableNames

Used to get or set the values of MySQL Server system variables.

Class
TMyServerControl

Syntax

```delphi
property Variables[const VarName: string]: string;
```

Parameters

VarName
- Holds the name of MySQL server system variables.

Remarks

Use the Variables property to get or set the values of MySQL Server system variables. String values should be assigned with quotes.

Example

For example:

```delphi
  MyServerControl1.Variables['max_allowed_packet'] := '1234567890';
  but:
  MyServerControl.Variables['time_format'] := ''''%H:%i:%s'''';
```
Methods of the `TMyServerControl` class.

For a complete list of the `TMyServerControl` class members, see the [TMyServerControl 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 <code>WHERE</code> clause of the <code>SELECT</code> statement in the SQL property.</td>
</tr>
<tr>
<td>AnalyzeTable</td>
<td>Analyzes and stores the key distribution for the table.</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>CheckTable</td>
<td>Checks MyISAM and InnoDB tables for failures.</td>
</tr>
<tr>
<td>CommitUpdates (inherited from <code>TMemDataSet</code>)</td>
<td>Clears the cached updates buffer.</td>
</tr>
<tr>
<td>CreateBlobStream (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>CreateDatabase</td>
<td>Creates a new database with the specified name.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</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>DropDatabase</td>
<td>Drops the specified database.</td>
</tr>
<tr>
<td>EditRangeEnd (inherited from TMemDataSet)</td>
<td>Enables changing the ending value for an existing range.</td>
</tr>
<tr>
<td>EditRangeStart (inherited from TMemDataSet)</td>
<td>Enables changing the starting value for an existing range.</td>
</tr>
<tr>
<td>Execute (inherited from TCustomDADataset)</td>
<td>Overloaded. Executes a SQL statement on the server.</td>
</tr>
<tr>
<td>Executing (inherited from TCustomDADataset)</td>
<td>Indicates whether SQL statement is still being executed.</td>
</tr>
<tr>
<td>Fetched (inherited from TCustomDADataset)</td>
<td>Used to find out whether TCustomDADataset has fetched all rows.</td>
</tr>
<tr>
<td>Fetching (inherited from TCustomDADataset)</td>
<td>Used to learn whether TCustomDADataset is still fetching rows.</td>
</tr>
<tr>
<td>FetchingAll (inherited from TCustomDADataset)</td>
<td>Used to learn whether TCustomDADataset is fetching all rows to the end.</td>
</tr>
<tr>
<td>FindKey (inherited from TCustomDADataset)</td>
<td>Searches for a record which contains specified field values.</td>
</tr>
<tr>
<td>FindMacro (inherited from TCustomDADataset)</td>
<td>Finds a macro with the specified name.</td>
</tr>
<tr>
<td>FindNearest (inherited from TCustomDADataset)</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 (inherited from TCustomDADataset)</td>
<td>Determines if a parameter with the specified name exists in a dataset.</td>
</tr>
<tr>
<td>Flush</td>
<td>Flushes server data from the</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetBlob</td>
<td>Overloaded. Retrieves TBlog 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>GetFieldEnum</td>
<td>Retrieve the list of acceptable values for a specified field given by the FieldName parameter.</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>GetServiceNames</td>
<td>Populates a list of MySQL services at the server to which Connection is connected.</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>KillProcess</td>
<td>Breaks the execution of the specified process.</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 MemDataSet.Locate method of TDataSet.</td>
</tr>
<tr>
<td>Function</td>
<td>Inherited From</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Lock</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>LockTable</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>MacroByName</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>OpenNext</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>OptimizeTable</td>
<td></td>
</tr>
<tr>
<td>ParamByName</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>Prepare</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>RefreshQuick</td>
<td>TCustomMyDataSet</td>
</tr>
<tr>
<td>RefreshRecord</td>
<td>TCustomDADataSet</td>
</tr>
<tr>
<td>RepairTable</td>
<td></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>ServiceStart</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ServiceStatus</strong></td>
<td>Returns a status of the specified service at the server to which Connection is connected.</td>
</tr>
<tr>
<td><strong>ServiceStop</strong></td>
<td>Stops the specified service at the server to which Connection is connected.</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>
<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>ShowProcessList</strong></td>
<td>Shows the list of the processes.</td>
</tr>
<tr>
<td><strong>ShowStatus</strong></td>
<td>Shows the current state of the server.</td>
</tr>
<tr>
<td><strong>ShowVariables</strong></td>
<td>Shows the values of some MySQL system variables that are in effect for the current connection.</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>UnLockTable</strong> (inherited from <strong>TCustomMyDataSet</strong>)</td>
<td>Releases a table locked by the TCustomMyDataSet.LockTable method.</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>
</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.

See Also
- **TMyServerControl Class**
- **TMyServerControl Class Members**

© 1997-2021 Devart. All Rights Reserved.

5.24.1.3.1 AnalyzeTable Method

Analyzes and stores the key distribution for the table

**Class**

**TMyServerControl**

**Syntax**

```plaintext
procedure AnalyzeTable;
```

**Remarks**

Call the AnalyzeTable method to analyze and store the key distribution for the table. During the analysis, the table is locked with a read lock. This works on MyISAM and BDB tables.

This is equivalent to running `myisamchk -a` on the table.

If the table hasn't changed since the last AnalyzeTable command, the table will not be analysed again.

The list of the tables is used from **TableNames**.

Refer to the ANALYZE TABLE detailed description at MySQL Reference Manual.

See Also
5.24.1.1.3.2 CheckTable Method

Checks MyISAM and InnoDB tables for failures.

Class

TMyServerControl

Syntax

procedure CheckTable(CheckTypes: TMyCheckTypes = [ctMedium]);

Parameters

CheckTypes

Holds the type of the checking that will be performed.

Remarks

Call the CheckTable method to check MyISAM and InnoDB tables for failures.

The list of the tables is used from TableNames.

Refer to the CHECK TABLE detailed description at MySQL Reference Manual.

See Also

- TableNames
- AnalyzeTable
- OptimizeTable
- RepairTable
5.24.1.1.3.3  CreateDatabase Method

Creates a new database with the specified name.

Class

TMyServerControl

Syntax

procedure CreateDatabase(DatabaseName: string; IfNotExists: boolean = True; CharsetName: string = ''; CollationName: string = '');

Parameters

DatabaseName
  Holds the name of the database that will be created.

IfNotExists
  True, if a database with the specified name does not exist.

CharsetName
  Holds the charset name.

CollationName
  Holds the collation name.

Remarks

Call the CreateDatabase method to create a new database with the specified name.

If IfNotExists is True, then a database will be created only if it did not exist.

CharsetName, CollationName parameters specify the language parameters of the created database.

See Also
• DropDatabase

© 1997-2021
Devart. All Rights Reserved.

5.24.1.1.3.4  DropDatabase Method

Drops the specified database.

Class
**TMyServerControl**

**Syntax**

```pascal
procedure DropDatabase(DatabaseName: string;IfExists: boolean = True);
```

**Parameters**

- **DatabaseName**
  - Holds the database name.
- **IfExists**
  - If a database with the specified name does not exist, an exception will be generated.

**Remarks**

Call the DropDatabase method to drop the specified database.

IfExists parameter is used to raise an error message if the specified database doesn't exist.

**Note:** Use this function very carefully!

**See Also**

- [DropDatabase](#)

© 1997-2021 Devart. All Rights Reserved.

---

**Flush Method**

Flushes server data from the memory to disk.

**Class**

**TMyServerControl**

**Syntax**

```pascal
procedure Flush(FlushTypes: TMyFlushTypes);
```

**Parameters**

- **FlushTypes**
  - Holds the options that define the actions to take while performing flush operation.

**Remarks**

Call the Flush method to flush forcefully server data from the memory to disk. Used if backup
5.24.1.3.6 GetServiceNames Method

Populates a list of MySQL services at the server to which Connection is connected.

Class

TMyServerControl

Syntax

procedure GetServiceNames(List: TStrings);

Parameters

List

Holds a list of MySQL services.

Remarks

Call the GetServiceNames property to populate a list of MySQL services at the server to which Connection is connected.

List.Objects[] fields are filled with the status of services in TMyServiceStatus format.

This method is available only for Windows.

Note: Any contents already in the target string list object are eliminated and overwritten by the data produced by GetServiceNames.

See Also

- ServiceStart
- ServiceStop
- ServiceStatus

© 1997-2021
Devart. All Rights Reserved.

© 2021 Devart
5.24.1.1.3.7 KillProcess Method

Breaks the execution of the specified process.

Class

TMyServerControl

Syntax

procedure KillProcess(ThreadId: integer);

Parameters

ThreadId

Holds the thread ID of the current connection.

Remarks

Call the KillProcess method to break the execution of the specified process.

Refer to KILL detailed description at MySQL Reference Manual.

See Also

- ShowProcessList

5.24.1.1.3.8 OptimizeTable Method

Packs tables deleting unused places from the file at the disk and defragmenting it.

Class

TMyServerControl

Syntax

procedure OptimizeTable;

Remarks

Call the OptimizeTable method to pack tables deleting unused places from the file at the disk and defragmenting it.

It makes sense to call it from time to time when working with a table actively, especially when
deleting frequently.
The list of the tables is used from `TableNames`.
Refer to `OPTIMIZE TABLE` detailed description at MySQL Reference Manual.

See Also

- `TableNames`
- `AnalyzeTable`
- `CheckTable`
- `RepairTable`

5.24.1.3.9 RepairTable Method

Repairs specified tables.

**Class**

`TMyServerControl`

**Syntax**

```plaintext
procedure RepairTable(RepairTypes: TMyRepairTypes = []);
```

**Parameters**

- `RepairTypes`
  
  Holds the options that define the actions to take while performing repair operation on a table.

**Remarks**

Call the RepairTable method to repair specified tables.

Works only with MyISAM tables.

List of the tables is used from `TableNames`.

Refer to `REPAIR TABLE` detailed description at MySQL Reference Manual.

See Also

- `TableNames`
- `AnalyzeTable`
5.24.1.1.3.10  ServiceStart Method

Starts the specified service at the server to which Connection is connected.

Class

TMyServerControl

Syntax

procedure ServiceStart(const ServiceName: string; ParamStr: string = '');

Parameters

ServiceName
  Holds the name of the service to start.

ParamStr
  Holds the list of parameters with which service will be started.

Remarks

Starts the specified service at the server to which Connection with ParamStr parameters is connected.

This method is available only for Windows.

See Also

• ServiceStop
• GetServiceNames
• ServiceStatus
5.24.1.1.3.11 ServiceStatus Method

Returns a status of the specified service at the server to which Connection is connected.

Class

TMyServerControl

Syntax

```pascal
function ServiceStatus(const ServiceName: string): TMyServiceStatus;
```

Parameters

ServiceName
Holds the name of the service to start.

Return Value
a status of the specified service.

Remarks

Call the ServiceStatus method to return a status of the specified service at the server to which Connection is connected.

To get a list of MySQL services you should use `GetServiceNames` function.

This method is available only for Windows.

See Also

- `ServiceStart`
- `ServiceStop`
- `GetServiceNames`

© 1997-2021
Devart. All Rights Reserved.

5.24.1.1.3.12 ServiceStop Method

Stops the specified service at the server to which Connection is connected.

Class

TMyServerControl

Syntax
procedure ServiceStop(const ServiceName: string);

Parameters

ServiceName
    Holds the name of the service to stop.

Remarks

Call the ServiceStop method to stop the specified service at the server to which Connection is connected.

This method is available only for Windows.

See Also

- ServiceStart
- GetServiceNames
- ServiceStatus

© 1997-2021 Devart. All Rights Reserved.

5.24.1.1.3.13 ShowProcessList Method

Shows the list of the processes.

Class

TMyServerControl

Syntax

procedure ShowProcessList(Full: boolean = False);

Parameters

Full
    True, if full query text will be shown.

Remarks

Call the ShowProcessList method to show the list of the processes. Full parameter specifies whether to show the full text of the query or only the first 100 symbols.

To disconnect use the KillProcess method.

5.24.1.1.3.14  ShowStatus Method

Shows the current state of the server.

Class

TMyServerControl

Syntax

procedure ShowStatus;

Remarks

Call the ShowStatus method to show the current state of the server.

See Also

• KillProcess

5.24.1.1.3.15  ShowVariables Method

Shows the values of some MySQL system variables that are in effect for the current connection.

Class

TMyServerControl

Syntax

procedure ShowVariables;

Remarks

Call the ShowVariables method to show the values of some MySQL system variables that are
in effect for the current connection.

See Also
- ShowStatus

## 5.25 MySQLApi

This unit contains implementation of the class.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyLogEvent</td>
<td>This type is used for the TMyEmbConnection.OnLog and TMyEmbConnection.OnLogError events.</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQLClientLibrary</td>
<td>When set, specifies path and name of MySQL client library (libmysql.dll or libmysqld.dll for embedded server).</td>
</tr>
</tbody>
</table>

## 5.25.1 Types

Types in the MySQLApi unit.

### Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMyLogEvent</td>
<td>This type is used for the TMyEmbConnection.OnLog</td>
</tr>
</tbody>
</table>
5.25.1.1 TMyLogEvent Procedure Reference

This type is used for the `TMyEmbConnection.OnLog` and `TMyEmbConnection.OnLogError` events.

Unit

`MySqlApi`

Syntax

```plaintext
TMyLogEvent = procedure (const Text: string) of object;
```

Parameters

`Text`

Holds the text of error, startup messages, entries that record client connections, and SQL statements.

5.25.2 Variables

Variables in the `MySqlApi` unit.

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MySQLClientLibrary</code></td>
<td>When set, specifies path and name of MySQL client library (libmysql.dll or libmysqld.dll for embedded server).</td>
</tr>
</tbody>
</table>
5.25.2.1 MySQLClientLibrary Variable

When set, specifies path and name of MySQL client library (libmysql.dll or libmysqld.dll for embedded server).

Unit
MySQLApi

Syntax

MySQLClientLibrary: string;

5.26 MySQLMonitor

This unit contains implementation of the TMySQLMonitor component.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMySQLMonitor</td>
<td>This component serves for monitoring dynamic SQL execution in MyDAC-based applications.</td>
</tr>
</tbody>
</table>

5.26.1 Classes

Classes in the MySQLMonitor unit.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMySQLMonitor</td>
<td>This component serves for monitoring dynamic SQL execution in MyDAC-based applications.</td>
</tr>
</tbody>
</table>

© 1997-2021 Devart. All Rights Reserved.

Request Support  DAC Forum  Provide Feedback
5.26.1.1 **TMySQLMonitor Class**

This component serves for monitoring dynamic SQL execution in MyDAC-based applications. For a list of all members of this type, see TMySQLMonitor members.

**Unit**

MySQLMonitor

**Syntax**

```plaintext
TMySQLMonitor = class(TCustomMySQLMonitor);
```

**Remarks**

Use TMySQLMonitor to monitor dynamic SQL execution in MyDAC-based applications. TMySQLMonitor 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 TMySQLMonitor object. If an application has no TMySQLMonitor instance, the Debug window is available to display SQL statements to be sent.

**Inheritance Hierarchy**

TCustomDASQLMonitor
  TCustomMySQLMonitor
    TMySQLMonitor
5.27 VirtualDataSet

This unit contains implementation of the TVirtualDataSet component.

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>

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>
**TCustomVirtualDataSet**
A base class for representation of arbitrary data in tabular form.

**TVirtualDataSet**
Dataset that processes arbitrary non-tabular data.

---

**5.27.1.1 TCustomVirtualDataSet Class**

A base class for representation of arbitrary data in tabular form.

For a list of all members of this type, see [TCustomVirtualDataSet](https://example.com) members.

**Unit**
**VirtualDataSet**

**Syntax**

```pascal
TCustomVirtualDataSet = class(TMemDataSet);
```

**Inheritance Hierarchy**

- **TMemDataSet**
- **TCustomVirtualDataSet**

---

**5.27.1.1.1 Members**

**TCustomVirtualDataSet** class overview.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CachedUpdates</td>
<td>(inherited from <strong>TMemDataSet</strong>) Used to enable or disable the use of cached updates for a dataset.</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>
</tbody>
</table>
### KeyExclusive (inherited from TMemDataSet)
Specifies the upper and lower boundaries for a range.

### LocalConstraints (inherited from TMemDataSet)
Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

### LocalUpdate (inherited from TMemDataSet)
Used to prevent implicit update of rows on database server.

### 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>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>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>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>
<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>
</tbody>
</table>
### 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>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>

---

### 5.27.1.2 **TVirtualDataSet Class**

Dataset that processes arbitrary non-tabular data.

For a list of all members of this type, see **TVirtualDataSet** members.

**Unit**

**virtualDataSet**

**Syntax**

```pascal
TVirtualDataSet = class(TCustomVirtualDataSet);
```

**Inheritance Hierarchy**
**TVirtualDataSet** class overview.

## 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>
</tbody>
</table>
### RevertRecord (inherited from **TMemDataSet**) unapplied.
Cancels changes made to the current record when cached updates are enabled.

### SaveToXML (inherited from **TMemDataSet**) Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

### 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.

### 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>OnUpdateError</td>
<td>Occurs when an exception is generated while cached updates are applied to a database.</td>
</tr>
</tbody>
</table>
OnUpdateRecord (inherited from TMemDataSet) | Occurs when a single update component can not handle the updates.

---

### 5.27.2 Types

Types in the VirtualDataSet unit.

**Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TOnDeleteRecordEvent</strong></td>
<td>This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.</td>
</tr>
<tr>
<td><strong>TOnGetFieldValueEvent</strong></td>
<td>This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.</td>
</tr>
<tr>
<td><strong>TOnGetRecordCountEvent</strong></td>
<td>This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.</td>
</tr>
<tr>
<td><strong>TOnModifyRecordEvent</strong></td>
<td>This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.</td>
</tr>
</tbody>
</table>

---

### 5.27.2.1 TOnDeleteRecordEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.

**Unit**

VirtualDataSet

**Syntax**

TOnDeleteRecordEvent = procedure (Sender: TObject; RecNo:
Type `TOnGetFieldValueEvent` is used for the `E:Devart.Dac.TVirtualDataSet.OnGetFieldValue` event.

**Unit**

`VirtualDataSet`

**Syntax**

```pascal
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.

---

**5.27.2.3 TOnGetRecordCountEvent Procedure Reference**

This type is used for the `E:Devart.Dac.TVirtualDataSet.OnGetRecordCount` event.

**Unit**

`VirtualDataSet`
Syntax

```delphi
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.

---

5.27.2.4 TOnModifyRecordEvent Procedure Reference

This type is used for `E:Devart.Dac.TVirtualDataSet.OnInsertRecord` and `E:Devart.Dac.TVirtualDataSet.OnModifyRecord` events.

**Unit**

`VirtualDataSet`

**Syntax**

```delphi
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.

---

5.28 VirtualTable

This unit contains implementation of the `TVirtualTable` component.

**Classes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

© 1997-2021 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback
### 5.28.1 Classes

Classes in the `VirtualTable` unit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TVirtualTable</code></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>

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**

```delphi
TVirtualTable = class(TMemDataSet);
```

**Inheritance Hierarchy**

- `TMemDataSet`
- `TVirtualTable`
**TVirtualTable** class overview.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>DefaultSortType</strong></td>
<td>Used to determine the default type of local sorting for string fields.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong> (inherited from <strong>TMemDataSet</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>Specifies the upper and lower boundaries for a range.</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 implicit update of rows on database server.</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>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>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<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>Assign</strong></td>
<td>Copies fields and data from another TDataSet component.</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>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></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</td>
</tr>
</tbody>
</table>
### MySQL Data Access Components

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 enabled.</td>
</tr>
<tr>
<td><strong>UpdateStatus</strong> (inherited from TMemDataSet)</td>
<td>Indicates the current update status for the dataset when cached updates are enabled.</td>
</tr>
</tbody>
</table>

### Events
## 5.28.1.1.2 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><strong>CachedUpdates</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Used to enable or disable the use of cached updates for a dataset.</td>
</tr>
<tr>
<td><strong>IndexFieldNames</strong></td>
<td>(inherited from <code>TMemDataSet</code>) 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 <code>TMemDataSet</code>) Specifies the upper and lower boundaries for a range.</td>
</tr>
<tr>
<td><strong>LocalConstraints</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening <code>TMemDataSet</code>.</td>
</tr>
<tr>
<td><strong>LocalUpdate</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Used to prevent implicit update of rows on database server.</td>
</tr>
<tr>
<td><strong>Prepared</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Determines whether a query is prepared for execution or not.</td>
</tr>
<tr>
<td><strong>Ranged</strong></td>
<td>(inherited from <code>TMemDataSet</code>) Indicates whether a range is applied to a dataset.</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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultSortType</td>
<td>Used to determine the default type of local sorting for string fields.</td>
</tr>
</tbody>
</table>

**See Also**
- TVirtualTable Class
- TVirtualTable Class Members

**Syntax**

```pascal
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 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>(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>Assign</strong></td>
<td>Copies fields and data from another TDataSet component.</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><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>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</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 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>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>SetRange</strong></td>
<td>Sets the starting and ending values of a range, and applies it.</td>
</tr>
<tr>
<td><strong>SetRangeEnd</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></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 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.

<table>
<thead>
<tr>
<th></th>
<th>enabled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateStatus</td>
<td>(inherited from TMemDataSet)</td>
</tr>
</tbody>
</table>

### See Also

- TVirtualTable Class
- TVirtualTable Class Members
LoadFromFile Method

Loads data from a file into a TVirtualTable component.

Class
TVirtualTable

Syntax
procedure LoadFromFile(const FileName: string; LoadFields: boolean = True; DecodeHTMLEntities: boolean = True);

Parameters
FileName
Holds the name of the file to load data from.

LoadFields
Indicates whether to load fields from the file.

DecodeHTMLEntities
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.

LoadFromStream Method

Copies data from a stream into a TVirtualTable component.

Class
TVirtualTable

Syntax
procedure LoadFromStream(Stream: TStream; LoadFields: boolean = True; DecodeHTMLEntities: boolean = True);

Parameters

Stream
Holds the stream from which the field's value is copied.

LoadFields
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.